HAW HAMBURG

BACHELORTHESIS
Alpha Oumar Diallo

Evaluierung Neuronaler Netze für die Objekterkennung auf Ein-Chip-Systemen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Alpha Oumar Diallo

Evaluierung Neuronaler Netze für die Objekterkennung auf Ein-Chip-Systemen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Marc Hensel
Zweitgutachter: Prof. Dr. Heike Neumann

Eingereicht am: 25. Februar 2022

Alpha Oumar Diallo

Thema der Arbeit

Evaluierung Neuronaler Netze für die Objekterkennung auf Ein-Chip-Systemen

Stichworte

Neuronale Netze, Objekterkennung, Verkehrsschild

Kurzzusammenfassung

Dieses Thesis behandelt die Evaluierung neuronaler Netze für ein Ein-Chip-System anhand der Erkennung von Verkehrsschildern in einem Modell-Fahrzeug. Die Arbeit achtet besonders auf die Architektur der Netzwerke für die Erkennung sowie die Performanz dieser Algorithmen.

Alpha Oumar Diallo

Title of Thesis

Evaluation of neural networks for object recognition on system-on-chip

Keywords

Neural networks, object recognition, traffic sign

Abstract

This thesis deals with the evaluation of neural networks for a one-chip system based on the recognition of traffic signs in model vehicles. The work pays particular attention to the architecture of the networks for the detection and performance of these algorithms.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

Tabellenverzeichnis **ix**

Abkürzungsverzeichnis **1**

1 Einleitung **2**
- 1.1 Motivation . . . 2
- 1.2 Zielsetzung . . . 3
- 1.3 Vorgehen und Struktur dieser Arbeit . . . 4

2 Grundlagen **5**
- 2.1 Künstliche Intelligenz Künstliche Intelligenz (KI) . . . 5
- 2.2 Machine Learning (ML) . . . 6
  - 2.2.1 Künstliches neuronales Netz . . . 7
  - 2.2.2 Muli-Layer Neuronale Netzwerk . . . 12
  - 2.2.3 Deep Learning . . . 13
  - 2.2.4 Convolutional Neural Networks (CNN) . . . 13
  - 2.2.5 Transfer Learning . . . 16
- 2.3 Programmierung und Tools . . . 17
  - 2.3.1 Bibliotheken . . . 17
  - 2.3.2 Programmiersprache . . . 20
- 2.4 Objekterkennung . . . 21
  - 2.4.1 Bounding Box . . . 22
  - 2.4.2 Objekterkennungsnetzwerke . . . 24
  - 2.4.3 Bilddatenbank . . . 29
- 2.5 Datensätze . . . 31
  - 2.5.1 Trainingsdaten . . . 31
  - 2.5.2 Validierungsdaten . . . 31

2.5.3 Testdaten . . . 32
2.6 Verkehrszeichen in Deutschland . . . 32
2.6.1 Gefahrzeichen (§40 stvo) . . . 32
2.6.2 Vorschriftszeichen (§41 stvo) . . . 32
2.6.3 Richtzeichen (§42 stvo) . . . 33

3 Anforderungsanalyse 35
3.1 Kontextabgrenzung . . . 35
3.1.1 Fachlicher Kontext . . . 35
3.1.2 Teschnischer Kontext . . . 36
3.2 Stakeholder . . . 37
3.3 Anwendungsfälle . . . 38
3.4 Funktionale und nicht-funktionale Anforderungen . . . 39
3.4.1 Funktionale Anforderungen . . . 39
3.4.2 Nicht-funktionale Anforderungen . . . 40

4 Konzept 41
4.1 Übersicht . . . 41
4.2 Übersicht des Hardware-Systems . . . 42
4.2.1 RC-Fahzeug . . . 43
4.2.2 Raspberry Pi . . . 43
4.2.3 Pi Kamera . . . 44
4.3 Auswahl des Netzwerkmodells . . . 45
4.4 Software . . . 48
4.4.1 Programmiersprache . . . 48
4.4.2 Entwicklungsumgebung . . . 49
4.4.3 Schnittstelle für Objekterkennungsmodell . . . 49
4.4.4 Extraktion von der maximalen Geschwindigkeit . . . 50
4.5 Datensatzaufbereitung . . . 50
4.5.1 Datensatzerzeugung . . . 51
4.5.2 Datensatzskaterosierung . . . 52
4.5.3 Annotierung des Datensatzes . . . 53
4.5.4 Aufteilung des Datensatzes . . . 55
4.6 Evaluationsmetriken . . . 55
4.6.1 Non Maximum Suppression . . . 56
4.6.2 Evaluationsmetriken . . . 56

4.7 Hyperparameter 59
4.8 Zusammenfassung 60

5 Umsetzung **61**
5.1 Datensatz 61
5.2 Training der Modelle 62
5.2.1 Yolov5s 62
5.2.2 You Only Look Once (YOLO)v4-Tiny 63
5.2.3 EfficientDet-D0 64
5.3 Extraktion der Geschwindigkeitsgrenze 65
5.4 Portierung zum Raspberry Pi 67

6 Auswertung **68**
6.1 Performanz der Modelle 68
6.1.1 Bildrate 68
6.1.2 Genauigkeit 69
6.2 Diskussion 69

7 Fazit **72**

Literaturverzeichnis **74**

A Anhang **80**

Selbstständigkeitserklärung **81**

Abbildungsverzeichnis

1.1 RC-Fahrzeug mit Raspberry Pi 4, Pi Kamera und Sender 3

2.1 Zusammenhang zwischen KI, Machine Learning (ML) und Deep Learning (DL), Bild wurde von [7, s. 5] abgeleitet. 6
2.2 Biologisches neuronales Netzwerk [5, s. 1] 7
2.3 Architektur eines Perzeptron [5, p. 5] 8
2.4 Sigmoid-Funktion [5, p. 13] 11
2.5 Verlauf von der Rectified Linear Units (ReLU)-Funktion [5, p. 13] 12
2.6 Tanh-Funktion [5, p. 13] 13
2.7 Architektur eines Feed-forward-Netzwerks mit zwei Hidden-Layer ohne bias [5, p. 18] 14
2.8 Struktur eines Convolutional Neural Networks (CNN)s [5] 15
2.9 Netzwerkarchitektur von YOLOv3 [15] 27
2.10 Netzwerkarchitektur von YOLOv3 [28] 28
2.11 Die Architektur von EfficientDet [54, p. 10784] 29
2.12 Gefahrzeichen: Gefahrstelle - Unebene Fahrbahn - Kurve (rechts) [2] . . . 33
2.13 Vorschrifstzeichen: Mindestgeschwindigkeit - Höchstgeschwindigkeit - Stop [2] 33
2.14 Richtzeichen: Vorfahrt - Fußgängerüberweg - Vorfahrtstraße [2] 34

3.1 Darstellung des Systems im Bezug des fachlichen Kontexts 36
3.2 Technische Interaktion zwischen das radio-controlled-car System (RCC-System) und den Beteiligten 37
3.3 Anwendungsfalldiagramme 38

4.1 Aktivitätsdiagramm des Systems 42
4.2 Raspberry Pi 4 B [44] 45
4.3 Raspberry Pi Camera Module 2 [43] 46
4.4 Arten von Objekterkennungsnetwerke laut Abschnitt 2.4.2 47

4.5 Workflow des Datensatzes 51
4.6 Aufgenommene Bilder mit Hilfe der Pi-Kamera 52
4.7 Angewendete Klassen von Verkehrschilder [2] 53
4.8 LabelImg Tool 55

5.1 Anzahl jeder klasse im Datensatz 62
5.2 Ablauf für die Extraktion der Geschwindigkeitsbregrenzung, wenn ein Vorschriftszeichen erkannt wurde 65
5.3 Koordinatensystem für das Zuschneiden des inneren Inhalts des Vorschriftszeichen. Im Bild Koordinaten ist die Y-Achse invertiert. 66

Tabellenverzeichnis

3.1 Stakeholder des Systems 38

4.1 Modell Vergleich [44] 44
4.2 Sensor Modes 44
4.3 Überblick über Modellen von *Model Zoo* [58] und YOLO [48] [8] [27] . . . 47
4.4 . Confusion matrix [19] 57
4.5 Angewandte Modellen für Objetkerkennung in dieser Arbeit 60

5.1 Trainingsergebnisse von YOLOv5s 63
5.2 Trainingsergebnisse des YOLOv4-Tiny Modells 64
5.3 Trainingserbnisse von EfficientDet-D0 64

6.1 Bildrate unter Raspberry Pi (RPI) 3B un 4B 69
6.2 Genauigkeit unter RPI 3B un 4B 69
6.3 Zustand der nicht-funktionalen Anforderungen 70
6.4 Zustand der nicht-funktionalen Anforderungen 70

Abkürzungsverzeichnis

CNN Convolutional Neural Networks.

DL Deep Learning.

FPS Frames Pro Sekunde.

IOU Intersection over union.

KI Künstliche Intelligenz.

mAP mean Average Precision.

ML Machine Learning.

NN Neuronales Netz.

RCC-System radio-controlled-car System.

RCNN Region-based Convolutional Neural Networks.

ReLU Rectified Linear Units.

ROI Region of Interest.

RPI Raspberry Pi.

SSD Single Shot Ein-Chip-Systemen.

YOLO You Only Look Once.

1 Einleitung

1.1 Motivation

Das menschliche Gehirn benutzt einen großen Anteil seiner Neuronen, um seine visuelle Wahrnehmung zu steuern [52]. Im Vergleich zu anderen ist das Sehorgan deutlich ausgeprägter als andere Sinne wie Fühlen und Hören. Das menschliche Gehirn verarbeitet die Daten extrem schnell und besitzt eine Datenbank an Vergleichsmustern [33]. Die Objekterkennung ist für das menschliche Gehirn eine Selbstverständlichkeit. Allerdings ist für eine Maschine diese Fähigkeit nicht vorhanden, denn sie hat am Anfang keine Vergleichsmuster und kann kein Objekt in einem Bild erkennen. Erst durch die Anwendung der KI, die auf ML basiert, bei dem Algorithmen so konfiguriert werden, dass sie ähnliche kognitive Funktionen verwenden wie das menschliche Gehirn, kann auch eine Maschine ähnliche Fähigkeiten erlernen. Anwendungen der Objekterkennung sind Kameras, die beim selbstfahrenden Auto die Hindernisse erkennen oder die adaptive Anpassung der Geschwindigkeit [20] durch die Verkehrsschildererkennung ermöglichen.

Heutzutage sind moderne Fahrzeuge in der Regel mit leistungsstarken Computern, die in der Lage sind, Daten von Sensoren und Kameras zu lesen und zu verarbeiten, ausgestattet. Um den Fahrer möglichst gut unterstützen zu können, werden nützliche Informationen zur Verfügung gestellt. Die Daten sollten schnell genug verarbeitet werden. Diese Systeme basieren typischerweise auf der Erkennung einer Region of Interest (ROI) [20] [24], in der sich das Verkehrszeichen befindet und typische Merkmale wie Farbe und geometrische Form erkannt werden. Diese Informationen liefern entscheidende visuelle Details, um die richtigen Fahrbedingungen zu verstehen. Sie informieren zum Beispiel über Geschwindigkeitsbegrenzungen, fahrbare Fahrspuren, oder Hindernisse.
Derzeit existieren bereits viele Frameworks zur Objekterkennung, die in Ansätzen zueinander unterscheiden. Ein anderer Aspekt ist die Verwendung von Deep-Learning-Modellen in kleinen Umgebungen. Das hilft um bessere Abschätzungen im Bezug der

Echtzeitmessungen. Um die Performenz des Modells auszuwerten, wird die Abschätzung der Echtzeitmessungen ein konkr.Aus diesem Grund war es der Wunsch von Professor Dr.-Ing. Marc Hensel die Anwendungen neuronaler Netze auf Ein-Chip-Systemen anhand der Erkennung von Verkehrschildern in Modell-Fahrzeugen (Abbildung 1.1) zu untersuchen.

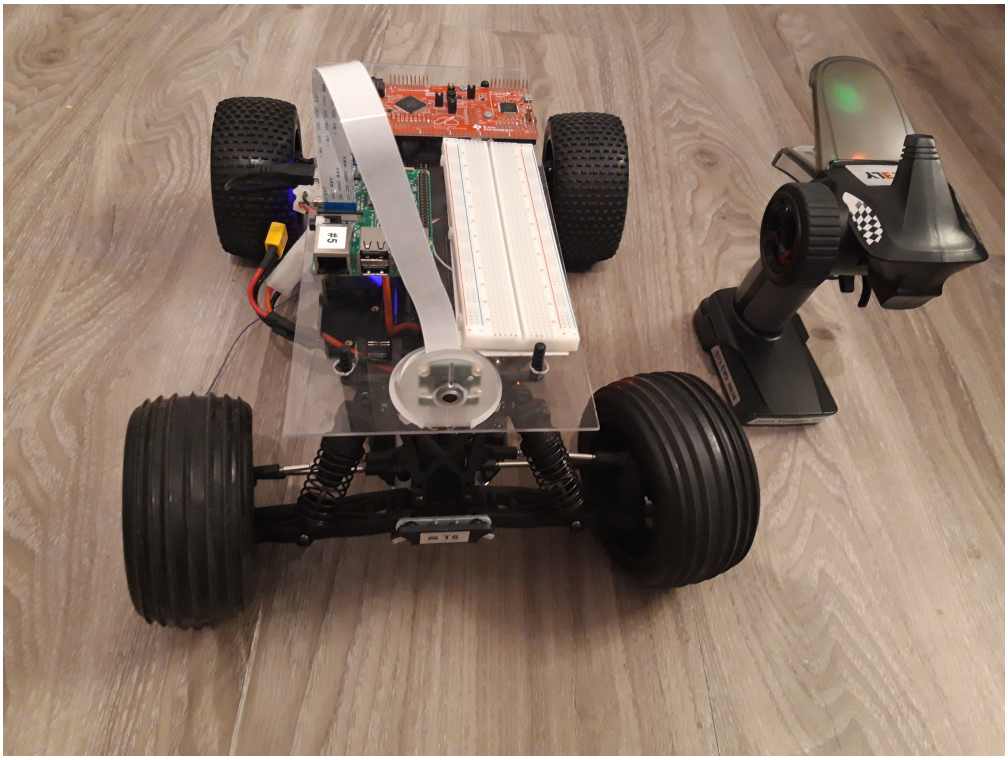


Abbildung 1.1: RC-Fahrzeug mit Raspberry Pi 4, Pi Kamera und Sender

1.2 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung eines Systems zur Erkennung von Geschwindigkeitsschildern auf einer RC-Fahrzeug-Plattform. Die Schilder sollen mit Hilfe einer Pi Kamera auf einem RPI erkannt werden. Hierzu werden künstliche neuronale Netze zur Erkennung der Schilder verwendet. Es werden drei neuronale Netze gewählt, trainiert und verglichen. Aus den Verkehrsschildern werden die Geschwindigkeitsschilder extrahiert. Über eine Schnittstelle wird dem RCC-System die aktuell erkannte Höchstgeschwindigkeit mitgeteilt. Der Schwerpunkt dieser Arbeit liegt auf der Auswahl von drei passenden neuronalen Netzen für ein Ein-Chip-System und deren Vergleich.

1.3 Vorgehen und Struktur dieser Arbeit

Die vorliegende Arbeit ist in sieben Teile unterteilt. Nach der Einleitung in Kapitel 1, werden in Kapitel 2 die nötigen Grundlagen erläutert. Im Kapitel 3 werden die Anforderung des Systems analysiert. Darüber hinaus wird in Kapitel 4 das Konzept aus den Anforderungen entwickelt. Daraufhin wird in Kapitel 5 die Umsetzung durchgeführt. In In Kapitel 6 wird die Anforderungsumsetzung aus Kapitel 3 ausgewertet und beurteilt . Abschließend wird in Kapitel 7 die Arbeit zusammengefasst und einen Ausblick auf weitere Entwicklungsmöglichkeiten gegeben.

2 Grundlagen

In diesem Kapitel werden die Grundlage vorgestellt, die zum Verständnis dieser Arbeit notwendig sind. Es werden zuerst die Themen KI und ML erläutert, dann die eingesetzten Programmiersprachen und Bibliotheken vorgestellt. Anschließend wird auf die Objekterkennung und ihre Herausforderungen eingegangen. Hier ist auch darauf hinzuweisen, dass alle Informationen in dem Abschnitt 2.2 auf [7] [55] und [5] beruhen, wenn nicht explizit referenziert wird.

2.1 Künstliche Intelligenz KI

Die KI ist die Fähigkeit eines Computers eine Aufgabe zu lösen, die normalerweise von Menschen gelöst wird [38]. Der Begriff wurde zum ersten Mal von John McCarthy im Jahr 1956 verwendet [7]. Ziel der KI ist es die biologische Intelligenz zu imitieren, um selbständig zu lernen oder zu agieren. Dies reduziert aktiv den Bedarf an manuellen, menschlichen Eingriffen für eine Vielzahl von Funktionen.
Generell wird KI als Bezeichnung für Computersysteme verwendet, die Aufgaben nach dem Training mit großen Datenmengen abarbeitet. In der Industrie ist die KI eine technologische Methode, die es ermöglicht, menschliche Wahrnehmungen und menschliches Handeln durch Maschinen nachzubilden. Die KI umfasst auch Teilbereiche des maschinellen Lernens und des DLs, die häufig in Verbindung mit künstlicher Intelligenz genannt werden. Diese Disziplinen bestehen aus KI-Algorithmen, die darauf abzielen, Systeme zu erstellen, die auf der Grundlage von Eingabedaten Vorhersagen oder Klassifizierungen treffen. In Abbildung 2.1 ist zu sehen, dass das DL ein Teilgebiet des MLs ist, wobei beides Teilgebiete der KI sind

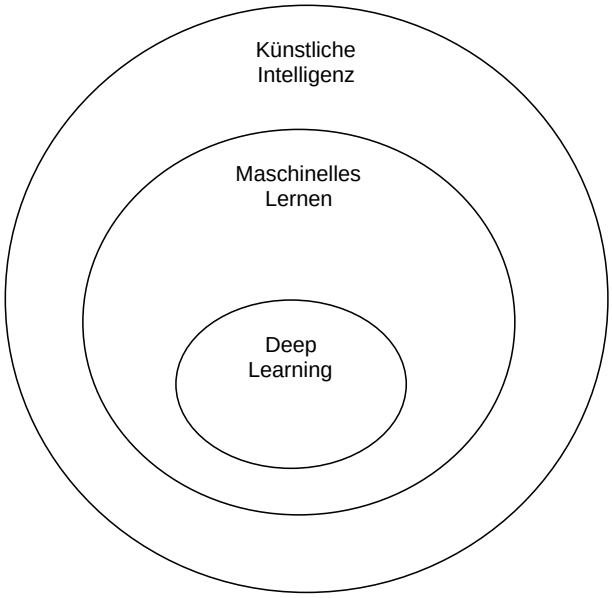


Abbildung 2.1: Zusammenhang zwischen KI, ML und DL, Bild wurde von [7, s. 5] abgeleitet.

2.2 Machine Learning (ML)

Das ML ist ein Teilgebiet der künstlichen Intelligenz, das mithilfe von Algorithmen aus Daten lernen kann und möglichst treffende Vorhersagen zu generieren. Das Ergebnis von ML-Algorithmen wird ML-Modell oder einfach Modell genannt. Nach einem erfolgreich abgeschlossenen Lernprozess wird das trainierte Modell dazu genutzt, unbekannte Daten zu bewerten. Auf diese Weise können ML-Algorithmen vorhersagen, indem sie auf der Grundlage früherer Erfahrungen Schlussfolgerungen und Entscheidungen treffen. Es kommt ohne explizite Programmierung oder ohne menschliches Eingreifen zu Schlussfolgerungen, indem es Muster identifiziert und vergangene Daten analysiert.
Es unterscheidet sich das Lernen je nach Art der Einlernphase. Während Datenpunkte der Trainingsdaten beim überwachten Lernen (Supervised Learning) gelabelt werden, werden beim unüberwachten Lernen (Unsupervised Learning) Datenpunkte der Trainingsdaten nach einem statistischen Modell nach Ähnlichkeit bestimmten Clustern zugeordnet, ohne dass diese explizit gelabelt wurden. Ein Beispiel für das unüberwachte Lernen ist die Empfehlung beim Online-Schoppen, Kunden die dieses Produkt gekauft haben, kauften auch jenes Produkt. Neben dem überwachten Lernen und dem unüberwachten Lernen

stellt Reinforcement Learning oder Bestärkungslernen die dritte Möglichkeit dar, Algorithmen so anzulernen, bei der ein Software-Programm (Agent) zu jedem Zeitschritt eine Aktion auswählen muss. Anschließend erhält der Agent möglicherweise eine Belohnung sowie neue Informationen über den Zustand der Umwelt. Eine Aktion wirkt sich dennoch oft erst nach vielen Zeitschritten aus und bewirkt dann eine positive oder negative Belohnung [38].

2.2.1 Künstliches neuronales Netz

Künstliche neuronale Netze sind beliebte Techniken des maschinellen Lernens, die den Mechanismus des Lernens in biologischen Organismen simulieren. Es ist ein System, das aus Neuronen besteht, die in der Regel in mehrere miteinander verbundene Schichten unterteilt sind. Das menschliche Nervensystem enthält Zellen, die als Neuronen bezeichnet werden. Die Neuronen sind unter Verwendung von Axonen und Dendriten miteinander verbunden, und die Verbindungsregionen zwischen Axonen und Dendriten werden als Synapsen bezeichnet. Diese Verbindungen sind in Abbildung 2.2 dargestellt. Die Stärken der Synaptik-Verbindungen ändern sich oft als Reaktion auf äußere Reize. Diese Veränderung ist die Vorgehensweise, wie Lernen in lebenden Organismen stattfindet.

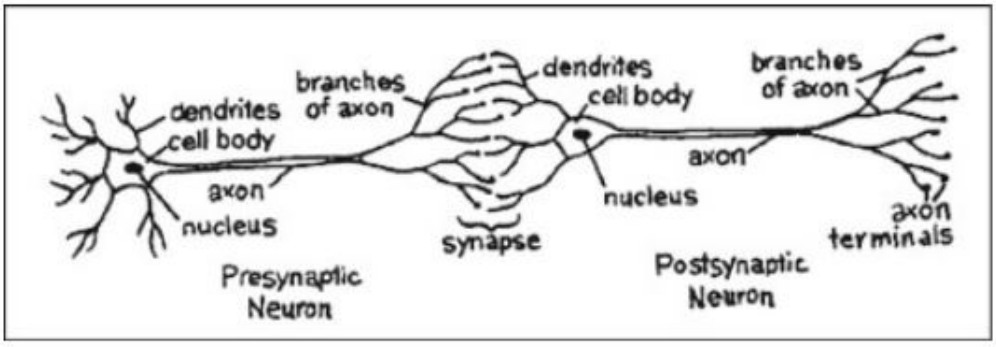


Abbildung 2.2: Biologisches neuronales Netzwerk [5, s. 1]

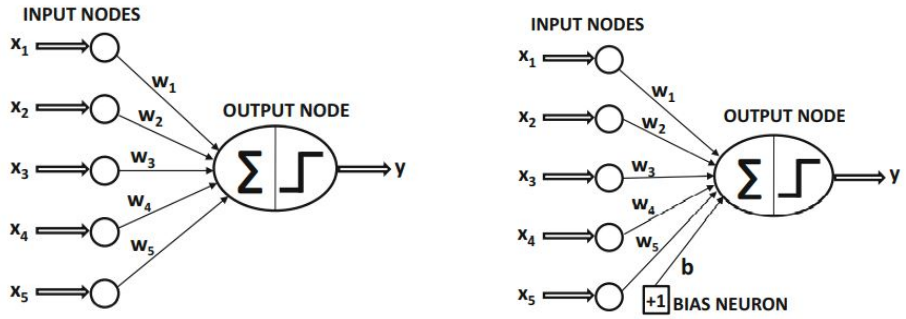
Im Prinzip sind neuronale Netze nichts anderes als eine Möglichkeit, parametrische Modelle zu bauen. Das heißt, dass die Entscheidungsfunktion explizit ist. Im Gegensatz zu anderen parametrischen Algorithmen wie der linearen Regression ermöglichen sie Ihnen die einfache Erstellung sehr komplexer und nichtlinearer Modelle. In dieser Arbeit wird

sich der Begriff **Neuronale Netze** auf künstliche neuronale Netze und nicht auf biologische Netze beziehen.

Das Perzeptron ist das einfachste mathematische Modell eines Künstlichen Neuronales Netz (NN)s. Im Jahr 1958 definierte *Frank Rosenblatt* erstmals das Perzeptron als Schicht von Neuronen, die in der einfachsten Form aus einem Ausgang und mehreren Eingängen besteht. Heutzutage spielt ihr biologischer Realismus keine große Rolle und es ist ihre Effizienz bei der Modellierung komplexer und nichtlinearer Zusammenhänge, die sie erfolgreich macht.

Perzeptron

Das einfachste neuronale Netzwerk, auch Perzeptron genannt, enthält eine einzelne Eingangschicht und einen Aufgabeknoten. Die grundlegende Architektur des Perzeptrons wird in Abbildung 2.3a gezeigt.



(a) Perzeptron ohne Bias (b) Perzeptron mit Bias

Abbildung 2.3: Architektur eines Perzeptron [5, p. 5]

Das Perzeptron ist ein Lernmodell, das für eine Eingabe $\overline{X}$ eine binäre Ausgabe y durch Auswerten einer Funktion $z = f(x)$ liefert. Das bedeutet, dass für das Erlernen von linearer Klassifizierbarkeit die Eingabewerte $[x_1, x_2, ..., x_d]$ in eine binäre parametrische Ausgabe y_i umgerechnet werden. Hierbei enthält jedes $[x_1, x_2, ..., x_d]$ Eingabenvariablen und z enthält die beobachteten Werte der binären Klassenvariablen. Als "beobachteter Wert"wird ein Wert bezeichnet, der als Teil der Trainingsdaten zur Verfügung gestellt wird. Dabei ist es das Ziel, die Klassenvariable für Fälle, in denen es nicht beobachtet wird, vorherzusagen. Die Gewichte $[W_1, W_2, ..., W_n]$ beziffern die Wichtigkeit der jeweiligen Eingabe für die Ausgangsparameter.

Die Eingabeschicht enthält d-Knoten, welche die d-Merkmale $[x_1, x_2, ..., x_d]$ mit Kanten des Gewichts $[W_1, W_2, ..., W_n]$ an einen Ausgabeknoten übertragen. Die Eingabeschicht führt keine eigene Berechnung durch. Die lineare Funktion $\overline{W}.\overline{X} = \sum_{j=1}^{d} w_j x_j$ wird am Ausgabeknoten berechnet. Die berechneten Werte werden in der Funktion $f()$ verwendet, um die abhängige Variable von $\overline{X}$ vorherzusagen. Daher wird die Vorhersage z wie folgt berechnet:

$$z = f(W \cdot X) = f(\sum_{j=1}^{d} w_j x_j), \tag{2.1}$$

wobei die Funktion f Aktivierungsfunktion genannt wird. Die Gleichung 2.1 zeigt, dass ein Neuron in der Tat zwei Funktionen innerhalb des Knotens berechnet. Zunächst wird $\sum_{j=1}^{d} w_j x_j$ berechnet, dann wird die Aktivierungsfunktion an dem Ergebnis verwendet. Die meisten der grundlegenden ML-Modelle können leicht als einfache neuronale Netzwerkarchitekturen dargestellt werden. Es ist eine nützliche Übung, traditionelle maschinelle Lerntechniken als neuronale Architekturen zu modellieren, da es ein klareres Bild davon vermittelt, wie Deep Learning das traditionelle maschinelle Lernen verallgemeinert. Dieser Standpunkt wird in Kapitel 2.2.3 detailliert betrachtet.

Bias

In vielen Situationen gibt es einen invarianten Teil der Vorhersage, der als **bias** (Verzerrung) bezeichnet wird. Bias ist ein Wert, der in der Vorwärtsberechnung jeder Schicht hinzugefügt werden kann. Es handelt sich um eine zusätzliche Verbindung zu jedem Knoten, die normalerweise auf eins gesetzt und nicht gewichtet ist. Mit der Bias b wird sich die Gleichung 2.1 ändern:

$$z = f(W.X + b) = f(\sum_{j=1}^{d} w_j x_j + b). \tag{2.2}$$

Auch in der Gleichung 2.2 ist z die Vorhersage und f die Aktivierungsfunktion. Das Bias kann als Gewicht einer Kante durch Verwendung eines Bias-Neurons integriert werden. Dies wird erreicht, indem ein Neuron hinzugefügt wird, das immer einen Wert von 1 an

den Ausgabeknoten überträgt. Das Gewicht der Kante, die das Bias-Neuron mit dem Ausgabeknoten verbindet, liefert die Bias-Variable. Ein Beispiel für ein Bias-Neuron ist in Abbildung 2.3b dargestellt.

Aktivierungsfunktion

Eine Aktivierungsfunktion f ist eine Funktion, die während der Vorhersage auf die Neuronen in einer Schicht angewendet wird. Die Funktion beschreibt, wie das Neuron auf das gewichtete Eingangssignal $a = w.x + b$ im Ausgangssignal $z = f(z)$ reagiert. Die Auswahl einer guten Aktivierungsfunktion liegen unter einigen Bedingungen:

- Die Funktion muss stetig sein und für alle Werte ihres Definitionsbereichs einen Funktionswert liefern. Das heißt, dass eine Aktivierungsfunktion für jede Eingabe eine Ausgabe liefern muss.
- Die zweite Bedingung besteht darin, dass gute Aktivierungsfunktionen monoton wachsend sein sollten und ihre Richtung nicht ändern dürfen. Anders formuliert muss der Funktionswert immer steigen.
- Als dritte Bedingung sollten gute Aktivierungsfunktionen nichtlinear sein. Aktivierungsfunktionen können nicht linear sein, da neuronale Netzwerke mit einer linearen Aktivierungsfunktion nur eine Schicht tief wirksam sind, unabhängig davon, wie komplex ihre Architektur ist. Eingaben in Netzwerke sind normalerweise lineare Transformation $W.X$, aber die reale Welt und Probleme sind nichtlinear.
- Eine gute Aktivierungsfunktion sollte sich effektiv berechnen lassen. Aktivierungsfunktionen werden sehr häufig aufgerufen, weshalb ihre Berechnung schnell sein sollte.

Hier werden Sigmoide-, ReLU- und Tanh-Aktivierungsfunktion vorgestellt.

Sigmoide Funktion ist eine mathematische Funktion und wird mit der Gleichung 2.3 beschrieben:

$$\theta(k) = \frac{\exp^5}{1 + \exp^u}. \tag{2.3}$$

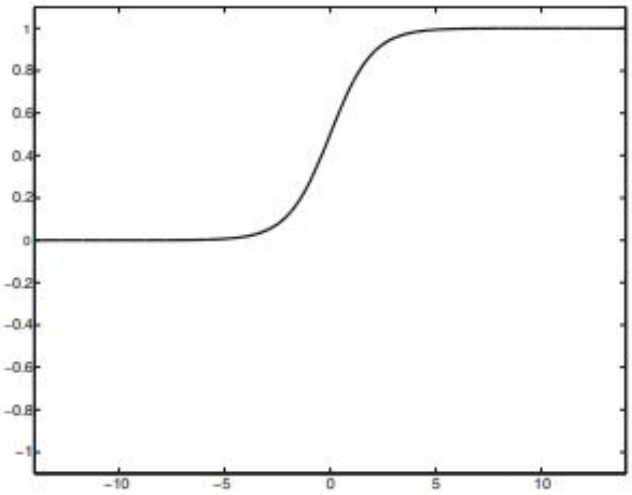


Abbildung 2.4: Sigmoid-Funktion [5, p. 13]

Die Sigmoid-Aktivierungsfunktion gibt einen Wert in $[0, 1]$ aus, was bei der Durchführung von Berechnungen hilfreich ist, die als Wahrscheinlichkeiten interpretiert werden sollten (Abbildung 2.4).

Rectified Linear Unit (ReLU) ist eine Aktivierungsfunktion, die eine nichtlineare Funktion oder stückweise lineare Funktion ist. Die ReLU gibt den Eingang direkt aus wenn er positiv ist, andernfalls wird Null ausgegeben. Die ReLU-Funktion ist durch die Gleichung 2.4 beschrieben und ihr Verlauf ist in Abbildung 2.5 dargestellt.

$$\Phi(k) = max(k, 0). \tag{2.4}$$

tanh ist eine Aktivierungsfunktion, die eine ähnliche Form wie die Sigmoidfunktion hat, mit der Ausnahme, dass sie horizontal neu skaliert und vertikal in $[-1, 1]$ skaliert wird. Die Funktion ist in der Abbildung 2.6 zu sehen. Die Tanh- und Sigmoid-funktion sind wie in der Gleichung 2.5 dargestellt miteinander verbunden.

$$tanh(u) = 2.sigmoid(2u) - 1. \tag{2.5}$$

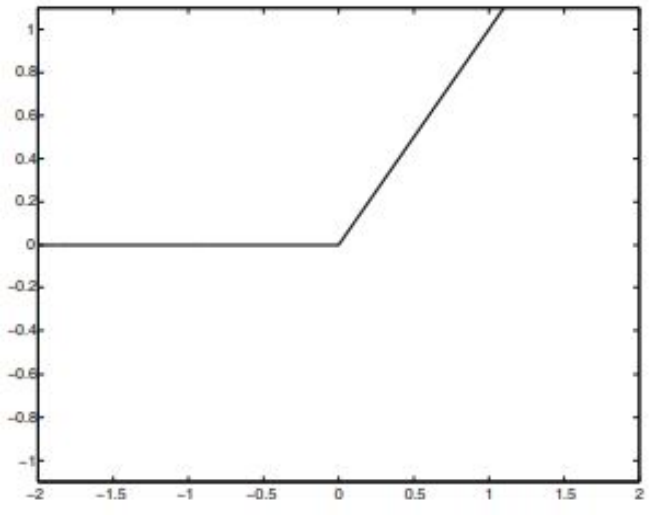


Abbildung 2.5: Verlauf von der ReLU-Funktion [5, p. 13]

2.2.2 Muli-Layer Neuronale Netzwerk

Ein Nachteil des Perzeptrons liegt in seiner linearen Trennung für eine Klassifikationsaufgabe, trotz der Verwendung einer nichtlinearen Aktivierungsfunktion. Diese Einschränkung kann umgangen werden, indem eine sogenannte verborgene Schicht (*engl. Hidden-Layer*) zwischen der Eingangschicht (*engl. Input-Layer*) und der Ausgangschicht (*engl. Output-Layer*) eingeführt wird. In der Fortsetzung dieser Arbeit werden die englische Begriffe für Eingang- Ausgangschicht und verborgene Schicht benutzt.
Muli-Layer neuronale Netzwerke enthalten mehr als 2 Layers. Das Perzeptron enthält einen Input- und einen Output-Layer, von denen der Output-Layer die einzige Rechenleistungsschicht ist. Die Eingabeschicht überträgt die Daten an die Ausgabeschicht und alle Berechnungen sind für den Benutzer vollständig sichtbar. Multi-Layer neuronale Netze enthalten mehrere Rechenschichten. Die zusätzlichen Zwischenschichten (zwischen Eingabe und Ausgabe) werden als Hidden Layer bezeichnet, da die durchgeführten Berechnungen für den Benutzer nicht sichtbar sind. Die spezifische Architektur mehrschichtiger neuronaler Netze werden als Feed-forward-Netze bezeichnet, da aufeinanderfolgende Schichten in Vorwärtsrichtung vom Eingang zum Ausgang ineinander speisen. Die Standardarchitektur von Feed-Forward-Netzwerken geht davon aus, dass alle Knoten in einer Schicht mit denen der nächsten Schicht verbunden sind. Die Abbildung 2.7 stellt ein Multi-Layer neuronales Netzwerk mit 2 Hidden-Layer ohne Bias dar.

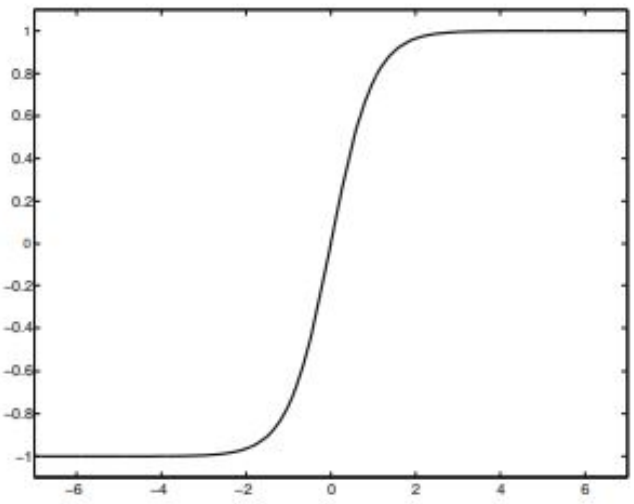


Abbildung 2.6: Tanh-Funktion [5, p. 13]

2.2.3 Deep Learning

Deep Learning auch tiefgehendes Lernen ist eine Teilmenge des ML. Die künstlichen neuronalen Netze verwenden beim Deep Learning eine komplexe innere Struktur, die durch eine Menge von Hidden-Layer zwischen Input-Layer und Output-Layer aufgebaut ist. Anders formuliert ist Deep Learning ein neuronales Netzwerk mit einer gewissen Komplexität, das ein neuronales Netzwerk mit mehr als zwei Schichten hat. Deep learning Netze verwenden raffinierte mathematische Modellierungen, um Daten auf komplexe Weise zu verarbeiten. Eine der wichtigsten Anwendungen dieser neuronalen Netze ist der Umgang mit unbeschrifteten oder unstrukturierten Daten, da Deep Learning versucht, die Fähigkeiten des Systems selbständig und ohne menschliche Hilfe zu verbessern. Das bedeutet, dass bei Deep Learning die Informationen für das Lernen bereitgestellt werden und die Analyse und das Ableiten von Prognosen oder Entscheidungen vom System des Deep Learnings sebst vorgenommen werden. Während bei ML der Mensch in die Analyse der Daten und in den Entscheidungsprozess eingreift.

2.2.4 Convolutional Neural Networks (CNN)

Ein CNN, auch ConvNet genannt, bezeichnet eine Unterkategorie von neuronalen Netzen und ist ein häufig eingesetzer Algorihmus für Deep Learning im Objekerkennungsbereich.

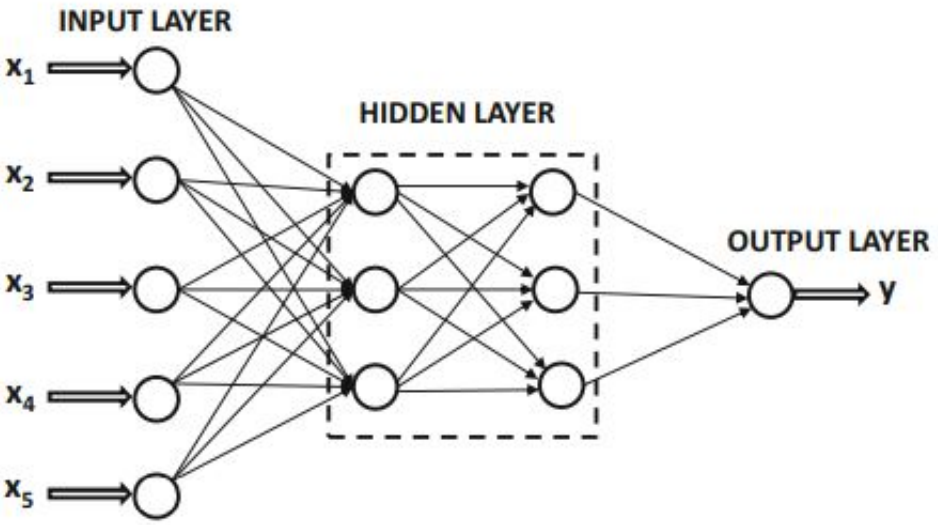


Abbildung 2.7: Architektur eines Feed-forward-Netzwerks mit zwei Hidden-Layer ohne bias [5, p. 18]

CNN sind vom Vorteil, um Muster in Bildern oder Szenen zu lokalisieren und anschließend zu erkennen. Die Stärke des CNN liegt darin, dass es dank Filtertechiken wichtige Merkmale wie Farben oder Formen und deren Beziehung zueinander erkennen kann.
Das Benutzen von CNN in Bildverarbeitung liegt in der Kapazität des Systems, sich die optimale Auswahl und Konfiguration von mehreren Filtern allein auf Basis der Trainigsdaten, die in der Bildverarbeitung aus Bildern besteht, selbst beizubringen. Die Struktur eines CNN bestheht aus einem Convolutional Layer, einem Pooling Layer und einem Fully-Connected-Layer und wird in der Abbildung 2.8 dargestellt.

Convolutional Layer

Die Faltungsschicht, auch Convolutional-Layer, ist die Schlüsselkomponente von CNN und bildet immer mindestens ihre erste Schicht. Sein Ziel ist es, das Vorhandensein einer Reihe von Merkmalen in den als Eingabe empfangenen Bildern zu identifizieren. Dazu führen wir eine Faltungsfilterung durch: Das Prinzip besteht darin, ein Fenster, welches das Merkmal darstellt, auf das Bild zu ziehen und das Faltungsprodukt zwischen dem Merkmal und jedem Teil des gescannten Bildes zu berechnen. Ein Merkmal wird dann als Filter betrachtet: Die beiden Begriffe sind in diesem Zusammenhang äquivalent. Die Faltungsschicht empfängt daher mehrere Bilder als Eingabe und berechnet die Faltung

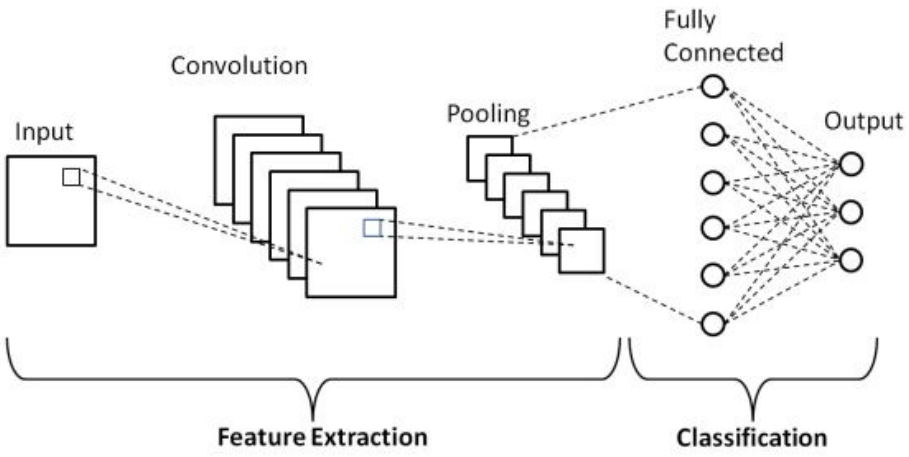


Abbildung 2.8: Struktur eines CNNs [5]

jedes von ihnen mit jedem Filter. Die Filter entsprechen genau den Merkmalen, die es in den Bildern finden wird.

Pooling Layer

Bei der Pooling-Operation geht es darum, die Größe der Bilder zu verkleinern und gleichzeitig deren wichtigsten Merkmale zu erhalten. Um dies zu erreichen, wird das Bild in regelmäßige Zellen zerlegt und dann innerhalb jeder Zelle der maximalen Wert behalten. In der Praxis werden oft kleine quadratische Zellen verwendet, um nicht zu viele Informationen zu verlieren. Die häufigsten Optionen sind benachbarte Zellen der Größe 2×2 Pixel, die sich nicht überlappen, oder Zellen der Größe 3×3 Pixel, die einen Schritt von 2 Pixeln voneinander entfernt sind, die sich also überschneiden. Die Pooling-layer reduziert die Anzahl der Parameter und Berechnungen im Netzwerk. Auf diese Weise wird die Effizienz des Netzes verbessert und Überlernen (Overfitting) vermieden.

Fully-connected-Layer

Der Fully-Connected-Layer besteht aus den Gewichten und Biases zusammen mit den Neuronen und wird verwendet, um die Neuronen zwischen zwei verschiedenen Layern zu verbinden. Dieser Layer wird normalerweise vor der Output-Layer verwendet und bildet

den letzten Layer einer CNN-Architektur. Dabei wird das Eingabebild aus den vorherigen Ebenen abgeflacht und der Fully-Connected-Ebene zugeführt. Der abgeflachte Vektor durchläuft dann einige weitere Fully-Connected-Layer, in denen die mathematischen Funktionsoperationen normalerweise stattfinden. In dieser Phase beginnt der Klassifizierungsprozess.

2.2.5 Transfer Learning

Das Transfer Learning ermöglicht das Übertragen der Ergebnisse eines fertig trainierten neuronalen Netzes auf eine neue Aufgabe. Das Prinzip ist, das von einem Netzwerk von Neuronen gewonnene Wissen bei der Lösung eines Problems zu nutzen, um ein anderes Problem zu lösen, das mehr oder weniger ähnlich ist. Auf diese Weise wird ein Transfer des Wissens durchgeführt.
Das Transfer Learning ist in der Praxis weit verbreitet und einfach zu implementieren. Es erfordert ein bereits vortrainiertes neuronales Netz, vorzugsweise auf einem Problem, das dem ähnelt, das wir lösen wollen. Heutzutage ist es leicht vortrainierte neuronale Netze aus dem Internet abzurufen, vor allem in den Bibliotheken von Deep Learning, wie Keras. Es gibt 3 Arten von Transfer Learning, die in den folgenden Abschnitten erklärt werden.

Fine Tuning von vortrainiertem Netzwerk

Für diese Variante des Transfer Learnings wird der letzte Fully-connected-Layer des vortrainierten Netzwerks durch einen an das neue Problem angepassten Klassifikator ersetzt und zufällig initialisiert. Alle Ebenen werden dann auf die neuen Bilder gezogen. Diese Variante sollte verwendet werden, wenn die neue Bilddatenbank groß ist. In diesem Fall ist das Risiko von Overfitting geringer. Da die Parameter aller Schichten, mit Ausnahme der letzten, zunächst die des vortrainierten Netzwerks sind, wird die Lernphase schneller durchgeführt, als wenn die Initialisierung zufällig gewesen wäre.

Vortrainiertes Netzwerk als Feature Extractor

Diese Variante besteht darin, die Funktionen des vortrainierten Netzwerks zu verwenden, um die Bilder des neuen Problems darzustellen. Der letzte Fully-connected-Layer wird entfernt und alle anderen Parameter werden festgelegt. Dieses abgeschnittene Netzwerk

berechnet somit die Darstellung jedes Eingangsbildes aus den bereits im Vortraining erlernten Merkmalen (features). Damit wird dann ein zufällig initialisierter Klassifikator auf diese Darstellungen trainiert, um das neue Problem zu lösen. Diese Variante sollte verwendet werden, wenn der Datensatz klein ist und den Bildern vor dem Training ähnelt. In der Tat ist es gefährlich, das Netzwerk auf so wenige Bilder zu ziehen, da das Risiko eines Overfittings wichtig ist. Wenn die neuen Bilder wie die alten aussehen, können sie außerdem durch die gleichen Merkmale dargestellt werden.

Vortrainiertes Netzwerk als Classifier

Hier wird der letzte Fully-connected Layer wieder durch den neuen zufällig initialisierten Klassifikator ersetzt und die Parameter bestimmter Schichten des vortrainierten Netzwerks festgelegt. So werden neben dem Klassifikator auch die neuen Bilder mit den unfixierten Schichten trainiert, die in der Regel den höchsten im Netzwerk entsprechen.

2.3 Programmierung und Tools

Um eine Aufgabe in NN zu lösen, werden Werkzeuge benötigt. In diesem Abschnitt werden zuerst die Bibliotheken vorgestellt, die in dieser Arbeit möglicherweise zum Einsatz kommen könnten. Anhand dieser Bibliotheken wird anschließend über die übliche Programmiersprache in NN diskutiert.

2.3.1 Bibliotheken

Es gibt viele Bibliotheken in ML. Hier werden welche vorgestellt, die häufig im Objekterkennungsbereich vorkommen.

OpenCV

OpenCV ist eine Open-Source-Computer-Vision-Bibliothek. Die Bibliothek ist in C und C++ geschrieben und läuft unter Linux, Windows, Mac OS X und mehr [10]. Es gibt eine aktive Entwicklung auf Schnittstellen für C++, Python, Java and MATLAB [9]. OpenCV wurde auf Recheneffizienz und mit einem starken Fokus auf Echtzeitanwendungen entwickelt [10]. Eines der Ziele von OpenCV ist es, eine einfach zu bedienende

Bildverarbeitungsinfrastruktur bereitzustellen, die Menschen hilft, ziemlich anspruchsvolle Bildverarbeitungsanwendungen schnell zu erstellen. Die OpenCV-Bibliothek enthält über 500 Funktionen, die viele Bereiche der Bildverarbeitung umfassen, darunter Werksproduktinspektion, medizinische Bildgebung, Sicherheit, Kamerakalibrierung, Robotik, Autonomes Fahren usw. [10]. Da die Bildverarbeitung und ML oft Hand in Hand gehen, enthält OpenCV auch eine vollständige, universelle Machine Learning Bibliothek. Diese Subbibliothek konzentriert sich auf die erkennung statistischer Muster und Clusterung. Die Machine Learning Bibliothek ist sehr nützlich für die Vision-Aufgaben, die im Mittelpunkt der Mission von OpenCV stehen, aber sie ist allgemein genug, um für jedes Problem der Maschinenverschleunigung verwendet zu werden [10].

TensorFlow

TensorFlow ist ein maschinelles Lernsystem, das in großem Maßstab und in heterogenen Umgebungen arbeitet [1]. Sein Rechenmodell basiert auf Datenflussgraphen mit veränderbarem Zustand. Graphknoten können verschiedenen Computern in einem Cluster und innerhalb jedes Computers CPUs, GPUs und anderen Geräten zugeordnet werden. TensorFlow unterstützt eine Vielzahl von Anwendungen, zielt aber insbesondere auf Training und Inferenz mit tiefen neuronalen Netzen ab. Es dient als Plattform für die Forschung und den Einsatz von maschinellen Lernsystemen in vielen Bereichen wie Spracherkennung, Bildverarbeitung, Robotik, etc. .

Keras

Keras ist eine kompakte und leicht zu erlernende High-Level-Python-Bibliothek für DL, die auf TensorFlow ausgeführt werden kann. Da es in Python geschrieben ist, hat es eine größere Gemeinschaft von Benutzern und Unterstützern und ist extrem einfach zu starten. Die Einfachheit von Keras besteht darin, dass es Benutzern hilft, DL-Modelle schnell zu entwickeln und eine Menge Flexibilität bietet, während es sich gleichzeitig um eine High-Level-API handelt. Dies macht Keras wirklich zu einem speziellen Framework. TensorFlow kann das Backend für Keras sein [34].

Caffe

Caffe ist eine vollständige Open-Source-Bibliothek, die ein sauberes und modifizierbares Framework für hochmoderne Deep-Learning-Algorithmen und eine Sammlung von Referenzmodellen bietet. Die in C++ geschriebene Bibliothek ist auch mit MATLAB und python implementiert [11] [26]. Caffe wird vom Berkeley Vision und Learning Center (BVLC) mit Hilfe einer aktiven Community von Mitwirkenden auf GitHub gepflegt und entwickelt. Es unterstützt laufende Forschungsprojekte, großtechnische Anwendungen und Startup-Prototypen in den Bereichen Bildverarbeitung oder Spracherkennung [26]. Darüber hinaus eignet sich der Code aufgrund seiner Modularität und der sauberen Trennung der Netzwerkdefinition von der eigentlichen Implementierung auch gut für den Forschungseinsatz. Modelle und Optimierungen können über die Einstellungsdatei ohne Codierung durchgeführt werden. Der Übergang zwischen CPU und GPU ist nahtlos. Aufgrund der hohen Verarbeitungsgeschwindigkeit kann es nützlich sein, es im Bildklassifizierungsproblem zu verwenden, das die Verarbeitung mit Millionen von Bildern erfordert [11].

Torch

Torch ist ein wissenschaftliches Berechnungsframework, das mit Lua erstellt wurde und auf einem Lua-Compiler (JIT) ausgeführt wird. Es verfügt über starke CUDA- und CPU-Backends und enthält gut entwickelte, ausgereifte Machine Learning- und Optimierungspakete. Die mitgelieferten Tensor-Bibliotheken verfügen über ein sehr effizientes CUDA-Backend und die Bibliotheken für neuronale Netze können verwendet werden, um beliebige azyklische Berechnungsgraphen mit automatischen Differenzierungsfunktionen zu erstellen [26].

Darknet

Darknet [46] ist ein Open-Source-Framework für neuronale Netzwerke, das in C und CUDA geschrieben ist. Es ist für seine hohe Geschwindigkeit und einfache Struktur bekannt. Leider unterstützt Darknet nur NVIDIA CUDA, um seine Berechnungen zu beschleunigen. Aus diesem Grund hat ein Benutzer nur begrenzte Möglichkeiten zur Grafikkartenauswahl [29].

TensorFlow Object detection

Die TensorFlow-Objekterkennungs-API ist ein Open-Source-Framework, das auf TensorFlow basiert und das Erstellen, Trainieren und Bereitstellen von Objekterkennungsmodellen vereinfacht. Es gibt bereits vortrainierte Modelle in ihren Schnittstellen, die als Model Zoo bezeichnet werden. Es enthält eine Sammlung vortrainierter Modelle, die auf verschiedenen Datensätzen trainiert wurden, wie z.B. COCO-Datensatz (Common Objects in Context) [58].

2.3.2 Programmiersprache

Im Abschitt 2.3.1 ist zu merken, dass Python, MATLAB und C++ die Programmiersprachen sind, die am häufigsten von den oben genannten Bibliotheken unterstützt werden. Aufgrund dieser Häufigkeit werden C++, Python und MATLAB hier präsentiert.

C++

C++ ist eine objektorientierte Programmiersprache mit einer schnelleren Laufzeit im Vergleich zu den meisten Programmiersprachen. Dies liegt daran, dass es näher an der Maschinensprache liegt. Aufgrund seiner Flexibilität und umfangreichen Bibliotheksunterstützung sowie einer aktiven Community sind viele Bibliotheken in der KI sowie ML Bereich mit implementiert [45].

Python

Python hat sich als eine der beliebtesten Sprachen für die Datenwissenschaft etabliert. Python verfügt über Bibliotheken zum Laden von Daten, Visualisierung, Statik, Bildverarbeitung und mehr. Diese umfangreiche Toolbox bietet Datenwissenschaftlern eine große Auswahl an allgemeinen und zweckmäßigen Funktionen. Einer der Hauptvorteile der Verwendung von Python ist die Möglichkeit, direkt mit dem Code zu interagieren, indem ein Terminal oder andere Tools wie das Jupyter Notebook verwendet werden. ML sind grundsätzlich iterative Prozesse, die zum Python-Konzept passen. [36].

MATLAB

MATLAB ist eine kommerzielle Software des Unternehmens Mathworks. Sein Zweck ist die Lösung mathematischer Probleme und die Repräsensatation der graphischen Ergebnisse [41]. Die Programmiersprache ist eine Script-sprache und ist vor allem für numerische Berechnung mit Matrizen geschafft. MATLAB hat bereits Tausende von ML-Anwendungen implementiert und vereinfacht ML mit Funktionen das Trainieren und Vergleichen von Modell oder Merkmalsextraktion.

2.4 Objekterkennung

Objekterkennung ist eine Computer-Vision-Technik, bei der ein Softwaresystem das Objekt aus einem bestimmten Bild oder Video erkennen, lokalisieren und verfolgen kann [4]. Die Besonderheit der Objekterkennung besteht darin, dass sie die Objektklasse (Person, Tisch, Stuhl etc.) und deren spezifische Koordinaten im gegebenen Bild identifiziert. Die Position wird angezeigt, indem ein Begrenzungsrahmen (*engl. Bounding Box* um das Objekt gezogen wird. Die Fähigkeit, das Objekt in einem Bild zu lokalisieren, definiert die Leistung des zur Erkennung verwendeten Algorithmus. Die Gesichtserkennung oder Verkehrsschilderkennung sind Beispiele für die Objekterkennung. Diese Objekterkennungsalgorithmen können vortrainiert sein oder können von Grund auf neu trainiert werden.
Traditionelle Objekterkennungstechniken folgen den 3 Hauptschritten.Im ersten Schritt werden mehrere ROI generiert. Diese ROI sind Kandidaten, die möglicherweise Objekte enthalten. Die Zahl dieser Regionen liegt in der Regel im Bereich von mehreren Tausend. Aus jedem ROI wird ein Merkmalsvektor fester Länge unter Verwendung verschiedener Bilddeskriptoren extrahiert. Dieser Merkmalsvektor ist entscheidend für den Erfolg der Objektdetektoren. Der Vektor sollte ein Objekt angemessen beschreiben, auch wenn es aufgrund einer Transformation wie Maßstab oder Translation variiert. Der Merkmalsvektor wird dann verwendet, um jeden ROI entweder der Hintergrundklasse oder einer der Objektklassen zuzuordnen. Mit zunehmender Anzahl von Klassen nimmt die Komplexität der Erstellung eines Modells zu, das zwischen all diesen Objekten unterscheiden kann. In diesem Abschinitt werden zunächst die Konzept Bounding-Box eläutert und einige Formate vorgestellt. Dann werden Objekterkennungsnetzwerke präsentiert. Anschießend wird auf einige Bilddatenbanken für die Objekterkennung eingagangen.

2.4.1 Bounding Box

Bounding Box, auch Begrenzungsrahmen genannt, ist ein imaginäres Rechteck, das als Bezugspunkt für die Objekterkennung dient und eine Kollisionsbox für dieses Objekt erzeugt [40]. Um Bilder zu annotieren (labeln) wird die sogenannte Bounding Box über die Bilder gezeichnet und das interessierende Objekt in jedem Bild durch Definieren seiner X- und Y-Koordinaten beschreiben. Dies erleichtert es Algorithmen des maschinellen Lernens, das Gesuchte zu finden, Kollisionspfade zu bestimmen und wertvolle Rechenressourcen zu schonen.
Um den Annotierungsprozess des Objekts in Bildern und Videos zu vereinfachen, wurden viele Tools entwickelt, um verschiedene Datensätze mit Anmerkungen zu versehen. Solche Werkzeuge bieten im Grunde die gleichen Features, wie z. B. Bounding-Box-Anmerkungen und polygonartige Silhouetten. Hier werden die Häufigsten Bounding-Box-Formate für Objekterkennung vorgestellt.

PASCAL-VOC-Format

Das PASCAL-VOC-Format besteht aus einer XML-Datei für jedes Bild, die eine oder mehrere Bounding-Boxes enthält [40]. Die Koordinaten einer Bounding-Box werden durch die Koordinaten der oberen linken und unteren rechten Ecke der Bounding-Box beschrieben und werden mit vier Werten in Pixel befasst:

$$(x_{min}, y_{min}, x_{max}, y_{max}).$$

Jede Bounding-Box enthält auch ein Etikett, das die Klasse des Objekts darstellt. Zusätzliche Informationen über das beschriftete Objekt können bereitgestellt werden, z. B. ob das Objekt über die Bounding-Box hinausreicht oder teilweise verdeckt ist. Die Annotierung in den Datensätzen ImageNet und PASCAL-VOC werden im PASCAL-VOC-Format bereitgestellt.

COCO-Format Das COCO-Format wird vom Common objects in Context (COCO)-Datensatz verwendet. Bei einer Annotierung in diesem Format werden alle Bounding-Boxes in einer einzigen JSON-Datei dargestellt [40]. Die Klassen der Objekte werden

in Kategorien separat aufgelistet und durch eine Identifiziert. Die Bilddatei, die einer Anmerkung entspricht, wird auch in einem separaten Element (images) angezeigt, das seinen Dateinamen enthält und durch eine Identiferenziert wird. Die Bounding-Box und ihre Objektklassen werden in einem anderen Element (Anmerkungen) mit ihren oberen linken Elementen aufgelistet. Die Bouding-Box wird in 4 Pixelwerte definiert:

$$(x_{min}, y_{min}, Breite, Hoehe).$$

TFRecord-Format Das TFRecord-Format ist eine serialisierte Darstellung des gesamten Datensatzes, der alle Bilder und Annotierungen in einer einzigen Datei enthält [40]. Dieses Format wird von der TensorFlow-Bibliothek erkannt.

TensorFlow Objekterkennung-Format Das Format verwendet ein eine CSV-Datei, die alle beschrifteten Bounding-Boxes des Datensatzes enthält. Dieses Format wird durch die Pixelkoordinaten oben links und unten rechts dargestellt [40]. Dieses Format ist auch ein weit verbreitetes Format, das von der TensorFlow-Bibliothek verwendet wird.

Yolo-Format In dieser Darstellung wird eine TXT-Datei pro Bild verwendet. Jede Zeile der Datei enthält die Kennzeichen für die Klassen und die Koordinaten der Bounding-Boxes [40]. Eine zusätzliche Datei ist erforderlich, um die Klassennamen und deren Kennzeichen zuzuordnen. Die Koordinaten der Bounding-Boxes sind durch das folgende Format gegeben:

$$(\frac{x_{center}}{Bildbreite}, \frac{y_{center}}{Bildhoehe}, \frac{Boxbreite}{Bildbreite}, \frac{Boxhoehe}{Bildhoehe}).$$

Der Vorteil der Darstellung der Kästchen in diesem Format besteht darin, dass sich bei Skalierung der Bildmaße die Koordinaten der Bouding Box nicht ändern und somit die Anmerkungsdatei nicht geändert werden muss [40]. Diese Art von Format wird von denjenigen bevorzugt, die Bilder in einer Auflösung kommentieren und ihre Abmessungen skalieren müssen, um die Eingabeformanforderungen eines bestimmten CNN zu erfüllen. Das YOLO-Objekterkennungsnetzwerk verwendet Bounding Boxes in diesem Format, um das Training durchzuführen.

2.4.2 Objekterkennungsnetzwerke

Das Objekterkennungsfeld hat sich sehr schnell entwickelt. Dank der kontinuierlichen Bemühungen vieler Forscher wachsen Objekterkennungsalgorithmen schnell, um die Leistung in Bezug auf der Genauigkeit als auch die Genauigkeit zu verbessern. In diesem Kapitel werden Netzwerke für die Objekterkennung vorgestellt. Aufgrund des weiten Feldes und verschiedenen Stand der Technik (*engl.* state-of-the-art) der Algorithmen ist es eine mühsame Aufgabe, alle Objekterkennungsnetzwerke in diesem Abschnitt abzudecken. *TensorFlow Detection Model Zoo* stellt eine Sammlung von Erkennungsmodellen zur Verfügung, die auf dem COCO 2017-Datensatz vortrainiert wurden und mit dem *TensorFlow Object Detection API* funktionieren [58]. Die Erkennungsmodelle werden als Familie präsentiert. Darüber hinaus wird die YOLO-Objekterkennung auch hier vorgestellt [48] [47] [8] [27].

Die CNN-Familie

Die CNN-Familie für die Objekerkennung umfasst Region-based Convolutional Neural Networks (RCNN) [57] Fast RCNN [21], Faster RCNN [50] und Mask RCNN [22]. Diese Objekterkennungsalgorithmen verwenden den ROI. Diese Technik besteht aus zweistufigen Detektoren, die den Detektionsfortschritt in der Generierung von Kandidatenboxen und die Erstellung von Vorhersagen laut den Kandidatenboxen unterteilen [18]. In diesem Abschnitt werden die oben genannten Algorithmen im Detail vorgestellt.

RCNN ist eine wichtige Etappe von CNN für die Objekterkennung. Es ist das erste neuronale Netzwerk, das den ROI vorschlägt, um die Objekterkennung basierend auf der hervorragenden Fähigkeit der Merkmalsextraktion und -klassifizierung von CNN zu realisieren. Der RCNN-Algorithmus verwendet eine Gruppe von Boxen für das Bild und analysiert dann in jeder Box, ob eine der Boxen ein Ziel enthält. Der ROI erhält die potenziellen Objekte durch Verschieben der Vorschläge mit unterschiedlicher Breite und Höhe, z. B. selektive Suche. Nach CNN werden Klassifizierung und Bounding-Box-Regression hinzugefügt, um das genaue Objekterkennungsergebnis zu erhalten [13] [3].

Fast RCNN verwendet ein Modell anstelle von drei verschiedenen CNN-Modellen, um Merkmale aus den verschiedenen Regionen zu extrahieren. Dann verteilt es die Regionen in mehrere Kategorien, basierend auf die aufzuweisenden Merkmale und die Grenzfelder der erkannten Divisionen kehren zusammen zurück. Fast RCNN verwendet die Methode des räumlichen Pyramidenpoolings (spatial pyramid pooling (SPP) [23]), um nur eine CNN-Darstellung für das gesamte Bild zu berechnen. Es übergibt eine Region für jedes Bild an ein bestimmtes Faltungsnetzwerkmodell, indem es drei verschiedene Modelle für den Auszug von Merkmalen, die Verteilung in Divisionen und die Erzeugung von Bounding-boxes ersetzt [3]. Diese neuen Methoden erhöhen die Genauigkeit des Algorithmus, indem sie die Klassifizierung und Regression kombinieren.

Faster RCNN Basierend auf Fast R-CNN löst Faster R-CNN [23] das regionale Vorschlagsproblem, indem es ein Region Proposal Network hinzufügt, was der Schlüsselbeitrag von Faster R-CNN ist. Es erhält den regionalen Vorschlag nicht auf dem Originalbild, sondern auf dem endgültigen Feature-Bild, das in das ROI-Pooling eingegeben wird. Da die Auflösung des Feature-Bildes niedriger ist als die des Originalbildes, ist die Berechnung von Faster R-CNN sicher viel geringer als die früheren Modelle von CNN.
Das Kernmerkmal des Region Proposal Networks besteht darin, den Vorschlag durch Verschieben zu erhalten. Jeder Schiebevorschlag generiert 9 Kandidatenanker mit unterschiedlichen Maßstäben, Breiten und Höhen. Der Verlauf von Faster R-CNN, die Merkmale der Anker zu extrahieren, ähnelt dem des Fast R-CNN, während seine Objektklassifizierung nur darin besteht, zu erkennen, dass die Merkmale im Vordergrund oder Hintergrund sind. Die Vorschlagsregression dient nur dazu, eine genauere Position des Zielobjekts herauszufinden. Für jede Position des Vorschlags verwendet RPN zwei vollständig verbundene Schichten (Objektklassifizierung und Vorschlagsregression), um die Anker zu beurteilen und aufzugeben.
Um alle Ziele in einem bestimmten Bild zu extrahieren, benötigt diese Prozedur mehrere Durchläufe für dieses bestimmte Bild. Verschiedene Systeme arbeiten in einer Reihenfolge, daher basiert die Leistung des bevorstehenden Vorgangs auf der Leistung der vorhergehenden Operationen. Bei diesem Ansatz werden Regionsvorschlagsnetzwerke verwendet, um die Objekte in einem Bild zu lokalisieren und zu identifizieren. Region Proposal Networks betrachten jedoch nicht das vollständige Bild, da sie nur die Teile des Bildes verwenden, die eine hohe Wahrscheinlichkeit für das Vorhandensein von Zielen aufweisen [3].

Mask RCNN wendet das gleiche, zweistufige Verfahren mit einer identischen ersten Stufe (RPN) an. In der zweiten Phase, parallel zur Vorhersage des Klassen- und Box-Offsets, gibt Mask RCNN auch eine binäre Maske für jeden ROI aus. Dies ist ein Unterschied zu den meisten neueren Systemen, bei denen die Klassifizierung von Maskenvorhersagen abhängt. Unser Ansatz folgt dem Geist von Fast R-CNN, das Bounding-Box-Klassifizierung und Regression parallel anwendet.

YOLO

YOLO ist ein hochmodernes Objekterkennungssystem, welches von Redmon u.a. im Jahr 2016 erstellt wurde und auf Echtzeitverarbeitung ausgerichtet ist [35]. YOLO wurde von GoogleNet inspiriert. Die Idee war, ein einzigartiges neuronales Netzwerk auf das vollständige Bild anzuwenden, wobei das Netzwerk das Bild in Regionen unterteilt und gleichzeitig Bouding-Box und Wahrscheinlichkeiten für jede Region vorhersagt. Diese Bouding-Boxen werden durch die vorhergesagten Wahrscheinlichkeiten gewichtet. YOLO teilt ein Bild in ein N $\times$ N-Raster auf, wobei jede Zelle nur ein Objekt vorhersagt. Diese Vorhersage wird als eine feste Anzahl von Begrenzungskästchen angegeben, wobei jedes Kästchen seinen Konfidenzwert hat. Es erkennt ein Objekt pro Rasterzelle unabhängig von der Anzahl der Kästchen, indem es einen Non-Maximum-Supprission-Algorithmus anwendet. YOLO verwendet im Allgemeinen ImageNet für das Parameter-Vortraining und verwendet dann Zielerkennungsdatensätze für das Zielerkennungstraining. Seit 2016 sind mehrere Verbesserungen der YOLO-Architektur erschienen. Während YOLOv1, YOLOv2 und YOLOv3 von Redmond entwickelt wurden, stammen YOLOv4 und YOLOv5 von 2 anderen Gruppen. Die Abbildung 2.9 zeigt eine vereinfachte Schichtstruktur von YOLOv3.

YOLOv3 verwendet eine Darknet-Architektur und hat 53 Schichten, die mit dem ImageNet-Datensatz trainiert wurden. Für die Objekterkennungsaufgaben wurden weitere 53 Schichten hinzugefügt und dieses Modell wurde mit dem Pascal VOC-Datensatz trainiert [48].

YOLOv4 wurde, anders als die drei früheren Versionen,von Alexey Bochkovskiy u.a. in 2020 entwickelt. YOLOv4 ist eine modifizierte Version von YOLOv3, und verwendet Cross Stage Partial Network (PSPNet) im Darknet [8].

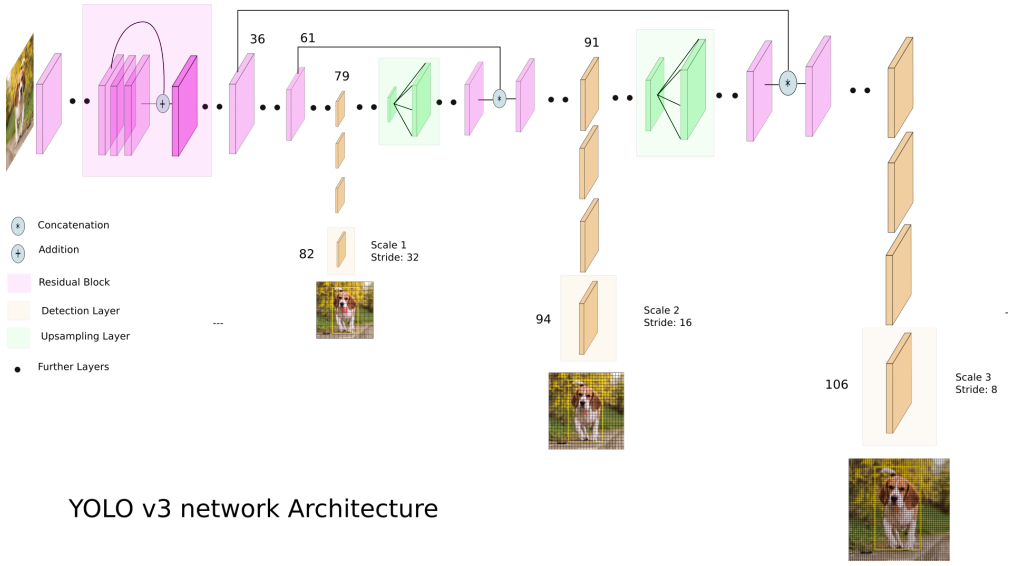


Abbildung 2.9: Netzwerkarchitektur von YOLOv3 [15]

YOLOv5 ist das neueste Objekterkennungsmodell, das von einem dritten Entwickler veröffentlichte wurde. Es wurde von ultralytics veröffentlicht und im Gegensatz zu allen anderen YOLO-Versionen ist es eine PyTorch-Implementierung [27].

Single Shot Ein-Chip-Systemen (SSD)

Das Single Shot MultiBox Detector (SSD)-Netzwerk wurde von Liu u.a. im Jahr 2015 veröffentlicht [35]. Das SSD-Netzwerk implementiert einen Algorithmus zum Erkennen mehrerer Objektklassen in Bildern durch Generieren von Konfidenzwerten einer beliebigen Objektkategorie in jeder Standardbox. Darüber hinaus werden Anpassungen in Kästchen vorgenommen, um die Objektformen besser anzupassen. Die SSD-Architektur ist CNN-basiert und zum Erkennen der Zielklassen von Objekten werden einerseits die Merkmale extrahiert und andererseits Faltungsfilter (*engl.* convolutional filters) zum Erkennen der Objekte angewendet.
Die Abbildung 2.10 zeigt, dass SSD VGG16 verwendet, um Merkmale zu extrahieren. VGG16 ist eine CNN-Architektur (Convolution Neural Net), die in 2014 den Imagenet-Wettbewerb gewonnen hat. SSD erkennt Objekte mithilfe der Conv4_3-Schicht von VGG16. Jede Vorhersage besteht aus einer Bouding-Box und 21 Punkten für jede Klasse, wobei es eine zusätzliche Klasse für kein Objekt gibt. Die Klasse mit der höchsten Punktzahl wird als diejenige für das begrenzte Objekt ausgewählt. Conv4_3 macht insgesamt

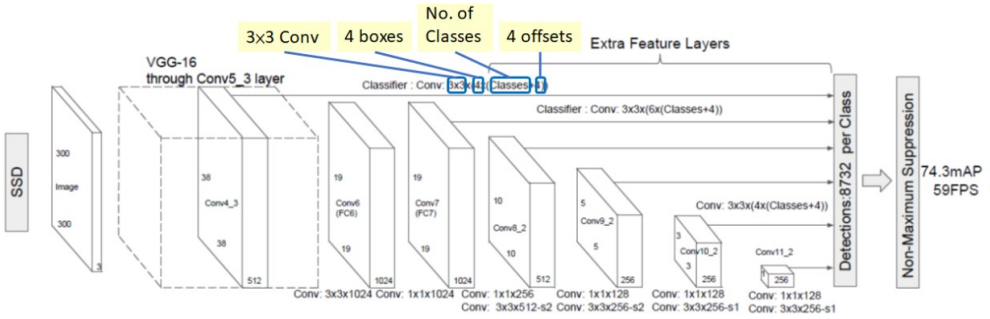


Abbildung 2.10: Netzwerkarchitektur von YOLOv3 [28]

$38 \times 38 \times 4$ Vorhersagen: Vier Vorhersagen pro Zelle, unabhängig von der Tiefe der Merkmalskarten. Viele Vorhersagen enthalten wie erwartet kein Objekt und verwenden die Klasse „0“, um anzuzeigen, dass kein Objekt im Bild erkannt wurde.

CenterNet

CenterNet ist ein einstufiges Objekterkennungsnetzwerk, das von Zhou u.a. veröffentlicht wurde [14]. Es verwendet zwei benutzerdefinierte Module namens Cascade Corner Pooling und Center Pooling, die die Rolle der Anreicherung von Informationen spielen. Die Informationen werden sowohl von der oberen linken, als auch von der unteren rechten Ecke gesammelt. Die jeweils besser erkennbare Information wird in der zentralen Region bereitgestellt. Die Idee dahinter ist, dass wenn eine vorhergesagte Bounding-Box einen hohen IoU mit der Ground-Truth-Box hat, dann ist die Wahrscheinlichkeit hoch, dass der zentrale Schlüsselpunkt (*engl.* keypoint) in seiner zentralen Region als dieselbe Klasse vorhergesagt wird und umgekehrt. Somit bestimmt das Netzwerk während der Schlussforgerung, nachdem ein Vorschlag als Paar von Eckschlüsselpunkten erzeugt wurde, ob der Vorschlag tatsächlich ein Objekt ist, indem geprüft wird, ob es einen mittleren Schlüsselpunkt derselben Klasse gibt, der in seinen zentralen Bereich fällt .

EfficientDet

Der EfficientDet-Algorithmus wurde vom *Google brain team* vorgeschlagen [54]. Das Netzwerk wurde auf ein früheres Netzwerk ËfficientNetäufgebaut, welches er als Backbone benutzt. Durch die Integration eines neuartigen bidirektionalen Funktionsnetzwerks (BiFPN) und neuer Skalierung erreicht EfficientDet eine Genauigkeit auf dem neuesten Stand

der Technik. Damit ist EfficientDet kleiner und braucht deutlich weniger Berechnungen im Vergleich zu Detektoren nach dem Stand der Technik. Die folgende Abbildung zeigt die gesamte Netzwerkarchitektur unserer Modelle. Dieses Backbone extrahiert Merkmale aus den Eingängen von P1 zu P7, wobei Pi die Merkmalsebene mit einer Auflösung von $1/2^i$ des Eingangsbildes darstellt. Wenn die Eingangsauflösung beispielsweise 640 x 640 beträgt, stellt P3in die Funktionsebene mit einer Auflösung von 80 x 80 dar .

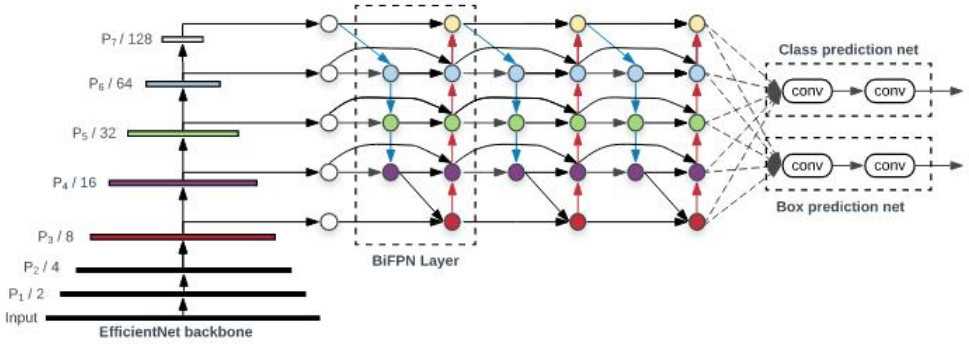


Abbildung 2.11: Die Architektur von EfficientDet [54, p. 10784]

.

2.4.3 Bilddatenbank

Hier werden Open-Source-Datensätze aufgelistet, die für Objekterkennungsprojekte verwendet werden können.

ImagenNet

Das ImageNet-Projekt ist eine große visuelle Datenbank, die für die Erforschung visueller Objekterkennungssoftware entwickelt wurde [12]. Mehr als 14 Millionen Bilder wurden vom Projekt von Hand gelabelt, um anzuzeigen, welche Objekte abgebildet sind und in mindestens einer Million der Bilder sind auch Begrenzungsrahmen vorgesehen. ImageNet enthält mehr als 20.000 Kategorien mit einer typischen Kategorie wie *Stuhl* oder *Tisch*, bestehend aus mehreren hundert Bildern. ImageNet enthält 1000 Kategorien für Objektlokalisierungen, 200 vollständig beschriftete Kategorien für die Objekterkennung oder auch 30 vollständig beschriftete Kategorien für Objekterkennungen aus Videos.

MS COCO

Das Microsoft Common Objects in Context (MS COCO) impliziert wörtlich, dass es sich bei den Bildern im Datensatz um Alltagsgegenstände handelt, die aus Alltagsszenen aufgenommen wurden. Das fügt den in den Szenen erfassten Objekten etwas Kontext hinzu [31]. COCO bietet Multi-Objekt-Beschriftungen, Segmentierungsmaskenanmerkungen, Bildunterschriften, oder Schlüsselpunkterkennung mit mehr als 328.000 Bildern und 91 Kategorien, was es zu einer sehr vielseitigen Datensatz macht.

ExDark

Exclusively DARK (ExDARK) ist ein einzigartiger Low-Light-Bilddatensatz, der eine Grundsammlung von Bildern für das Benchmarking von Forschungsarbeiten bei schlechten Lichtverhältnissen bietet und verschiedene Fachgebiete zusammenbringt, um sich auf Bedingungen bei schlechten Lichtverhältnissen zu konzentrieren, z. B. Bildverständnis, Bildverbesserung oder Objekterkennung [32]. Der Datensatz ist eine Sammlung von 7.363 Bildern bei schlechten Lichtverhältnissen von Umgebungen mit sehr wenig Licht bis zur Dämmerung mit 12 Objektklassen, die sowohl auf Bildklassenebene als auch auf lokalen Objektbegrenzungsrahmen mit Anmerkungen versehen sind.

LISA Traffic Sign Detection Dataset

LISA oder Laboratory for Intelligent & Safe Automobiles Verkehrsschild-Datensatz ist eine Reihe von Videos und kommentierten Frames, die US-Verkehrszeichen enthalten [37]. Es wird in zwei Stufen veröffentlicht, eine mit nur den Bildern und eine mit Bildern und Videos. Der Datensatz enthält Bilder von verschiedenen Kameras, 47 US-Zeichentypen und 7855 Annotierungen auf 6610 Bilder.

Open Images

Open Images ist ein Datensatz von rund 9 Millionen Bildern, die mit Beschriftungen auf Bildebene, Objektbegrenzungsrahmen, Objektsegmentierungsmasken, visuellen Beziehungen und lokalisierten Erzählungen kommentiert wurden [30]. Der Datensatz enthält 16 Millionen Begrenzungsrahmen für 600 Objektklassen auf 1,9 Millionen Bildern und ist damit der größte vorhandene Datensatz mit Objektpositionsanmerkungen. Die Boxen

wurden größtenteils manuell von professionellen Annotatoren gezeichnet, um Genauigkeit und Konsistenz zu gewährleisten. Open Images bietet auch visuelle Beziehungsanmerkungen, die Paare von Objekten in bestimmten Beziehungen, Objekteigenschaften und menschlichen Aktionen anzeigen.

Pascal VOC-Datensatz

PASCAL Visual Object Classes (VOC) ist ein sehr beliebter Datensatz zum Erstellen und Bewerten von Algorithmen für die Bildklassifizierung, Objekterkennung und Segmentierung [16]. Der PASCAL VOC 2012 Datensatz enthält 20 Objektkategorien, darunter Fahrzeuge oder Haushalt und enthält 11.530 Bilder für Trainings- und Testdaten, 27.450 -annotierten Bilder und 6.929 Segmentierungen.

2.5 Datensätze

Um ein Machine Learning Modell korrekt zu trainieren wird ein Datensatz benötigt [55]. Bei überwachten Lernverfahren wird dieser Datensatz in der Regel in drei verschiedene Datensätze unterteilt:

2.5.1 Trainingsdaten

Die Trainingsdaten sind eine Teilmenge des gesamten Datensatzes. In der Regel wird kein Modell auf der Gesamtheit der Daten trainiert. Es ist die Datenprobe, die zur Anpassung des Modells verwendet wird. Das Modell sieht und lernt aus diesen Daten.

2.5.2 Validierungsdaten

Die Validierungsdaten sind die Stichprobe von Daten, die verwendet werden, um eine unverzerrte Bewertung einer Modellanpassung an den Trainings-Datensatz bereitzustellen, während die Hyperparameter des Modells optimiert werden. Die Bewertung wird voreingenommener, wenn die Kenntnisse des Validierungs-Datensatzes in die Modellkonfiguration integriert werden. Die Validierungdaten werden verwendet, um ein bestimmtes Modell zu bewerten. Dies ist jedoch für häufige Auswertungen.

2.5.3 Testdaten

Die Datenstichprobe, die verwendet wird, um eine unverzerrte Bewertung einer endgültigen Modellanpassung an den Trainingsdatensatz bereitzustellen werden Testdaten geanennt. Der Test-Datensatz bietet die entsprechende Vorgehensweise, die zur Bewertung des Modells verwendet wird. Dieser wird erst verwendet, wenn ein Modell vollständig trainiert ist. Die Testdaten werden im Allgemeinen verwendet, um konkurrierende Modelle zu bewerten.

2.6 Verkehrszeichen in Deutschland

Als Teil der Straßenausstattung dienen die Verkehrszeichen der Verkehrsregelung und werden behördlich angeordnet [53] [17]. Laut der deutschen Straßenverkehrs-Ordnung, im § 39 Abs. 2 Satz 2 StVO, sind die Verkehrszeichen in 3 Kategorien Gruppiert: Gefahrzeichen, Vorschriftszeichen und Richtzeichen.

2.6.1 Gefahrzeichen (§40 stvo)

Die Gefahrzeichen weisen den Verkehrsteilnehmer zu erhöhter Aufmerksamkeit hin, wenn dieser sich einer potentiellen Gefahrstelle nähert. Diese Vehrkehrszeichen sind in einer dreieckigen Form mit rotem Rand. Dennoch zählen auch verschiedene rotweiß gestreifte Baken, wie zum Beispiel an Bahnübergängen, zu ihnen. Ein anderes Beispiel für Gefahrzeichen ist das Gefahrstellenschild für Gefahren jeglicher Art. Mit einer Verringerung der Geschwindigkeit ist zu rechnen oder eine Bremsbereitschaft muss gegeben sein. Dazu Zählen auch die Kurve (rechts oder links) für die Warnung vor einer Scharfen Rechtskurve bzw. Linkskurve, in die man nicht schnell hineinbiegen sollte oder ein unebene Fahrbahn-Schild für eine Reduzierung der Geschwindigkeit. Wegen der Unebenheit der Fahrbahn kann es zu Schäden am Fahrzeug oder der Ladung des Fahrzeuges führen. Die Abbildung 2.12 stellt drei verschiedenen Gefahrzeichen dar.

2.6.2 Vorschriftszeichen (§41 stvo)

Die Vorschriftszeichen fassen Gebote und Verbote. Sie sind meist in runder Schilderform und können durch Zusatzschilder ergänzt und (eventuell) beschränkt werden. Zu

Abbildung 2.12: Gefahrzeichen: Gefahrstelle - Unebene Fahrbahn - Kurve (rechts) [2]

den Vorschriftszeichen gehören die Geschwindigkeitsbegrenzungen, die Hinweise zu vorgeschriebenen Fahrtrichtungen sowie die Verbote für Verhaltensweisen. Dazu werden auch die weißen Markierungen auf den Straßen, wie Zebrahstreifen, Fahrstreifenbegrenzung, Leitlinie und Haltlinie gezählt. Beispiele folgen in der Abbildung 2.13.

Abbildung 2.13: Vorschrifstzeichen: Mindestgeschwindigkeit - Höchstgeschwindigkeit - Stop [2]

2.6.3 Richtzeichen (§42 stvo)

Die Richtzeichen weisen auf besondere Hinweise zur Erleichterung des Verkehrs hin. Sie können sowohl Gebote als auch Verbote enthalten. Sie gelten unmittelbar. Das Vorfahrtszeichen ist ein Richtzeichen, das dem Verkehrsteilnehmer nur an der nächsten Kreuzung oder Einmündung Vorfahrt gewährt. Diese Schilder stehen in geschlossenen Ortschaften unmittelbar vor der Kreuzung bzw. Einmündung. Außerhalb geschlossener Ortschaften steht es normalerweise 150 bis 250 m davor. Ein zweites Beispiel eines Richtzeichens ist das Zeichen für einen Fußgängerüberweg. Dieses Schild betont den Fahrer besondere Vorsicht und Aufmerksamkeit auf Fußgänger, die die Straße Überqueren möchten. In diesem Fall muss der Fahrer das Auto anhalten und den Fußgänger die Überquerung der Straße gewähren. Die Vorfahrtsstraße ist ein Vorfahrtsschild, welches für alle folgenden Kreuzungen und Einmündungen innerhalb geschlossener Ortschaften gilt, bis durch eines der Zeichen "Vorfahrt gewähren", "Halt Vorfahrt gewährenöder auch Ende

der "Vorfahrtsstraße"die Vorfahrt eingestellt wird. Außerhalb geschlossener Ortschaften wird dieses Vorfahrtsschild als Parkverbot auf der Fahrbahn eingesetzt. Die Abbildung 2.14 zeigt Beispiele von Richtzeichen.

Abbildung 2.14: Richtzeichen: Vorfahrt - Fußgängerüberweg - Vorfahrtstraße [2]

3 Anforderungsanalyse

Die Anforderungsanalyse ist ein Prozess, mit dem die Erfordernisse festgestellt werden, denen ein System für eine bestimmte Aufgabenstellung gerecht werden sollte. Auch wenn die eigentliche Hardware-Entwicklung nicht Teil dieser Arbeit ist, wird das Gesamtsystem, beziehungsweise die Hardware, welche das Auto, den Sender sowie die Kamera und das Ein-Chip-System einschließt, des Weiteren als RCC-System bezeichnet. Zunächst wird die Kontextabgrenzung vorgestellt, in welcher das System und die Systemgrenze definiert wird. Es werden danach die Stakeholder und Beteiligten analysiert und präsentiert. Dazu werden die Anwendungsfälle beschrieben. Anschließend werden die Anforderungen an das System formuliert.

3.1 Kontextabgrenzung

Die Kontextabgrenzung definiert das System und seine Grenze. Außerdem legt sie den Inhalt des Systemkontexts fest und grenzt das System von Nachbarsystemen ab. Hier werden auf den fachlichen Kontext und den technischen Kontext im Detail eingegangen.

3.1.1 Fachlicher Kontext

In diesem Abschnitt geht es darum, eine Entscheidung darüber zu treffen, ob der Aspekt, den man betrachtet, im Kontext des Systems steht oder für das geplante System nicht relevant ist. Die Abbildung 3.1 gibt einen Überblick der Systemdarstellung im fachlichen Kontext.

Verkehrsschild Die Verkehrsschilder dienen als Objekttypen für die Erkennung. Es basiert auf die deutschen Verkehrsschilder.

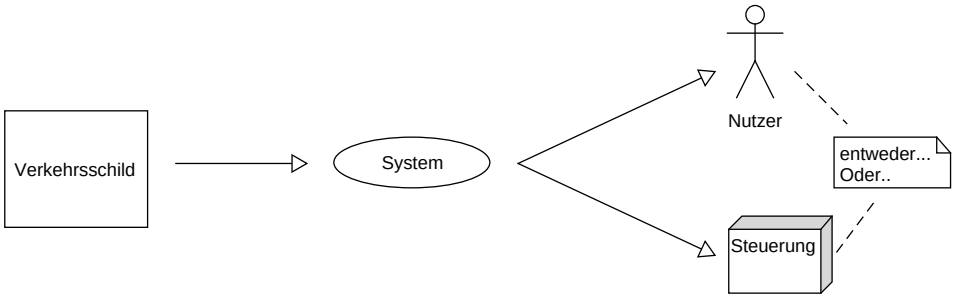


Abbildung 3.1: Darstellung des Systems im Bezug des fachlichen Kontexts

System Das System oder auch RCC-System genannt ist ein Kompositum von Bauteilen, welches unter anderen RC-Fahrzeug, Kamera und weitere Bauteile wie Elektronik oder auch Kabel enthält. Das RCC-System verarbeitet die aufgenommenen Daten der Kamera und übermittelt dem Nutzer die Informationen.

Nutzer Der Nutzer Steuert das RC-Fahrzeug manuell und wird vom RCC-System benachrichtigt, wenn eine bestimmte Anforderungen erfüllt wird.

Steuerung Alternativ zum Nutzer kann auch die Steuerung vom RCC-System benachrichtigt werden. Die Anforderungen bezüglich der Informationsbenachrichtigungen sind identisch.

3.1.2 Teschnischer Kontext

Der technische Kontext beschreibt die technischen Schnittstellen zwischen dem System und seinen Nachbarsystemen. Zusätzlich gibt der technische Kontext einen Überblick, welche fachlichen Ein- und Ausgaben über welche technischen Schnittstellen fließen. Die Abbildung 3.2 stellt den technischen Kontext des Systems dar.

Da die Kamera-Schnittstelle kein erforderlicher Teil für diese Arbeit ist, werden die technischen Schnittstellen Videoverarbeitungsprogramm und Objekterkennungsmodell erläutert.

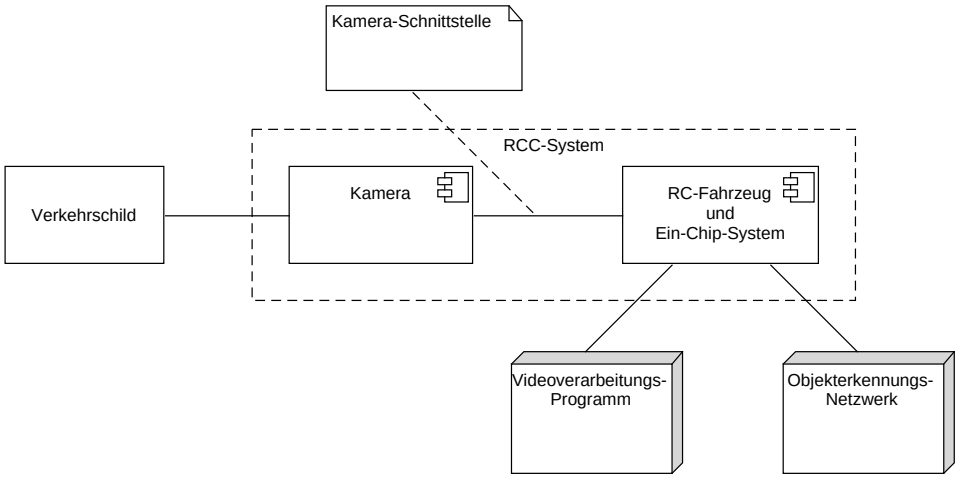


Abbildung 3.2: Technische Interaktion zwischen das RCC-System und den Beteiligten

Videoverarbeitungsprogramm Das Programm welches auf dem Ein-Chip-System läuft, versucht Verkehrsschilder, mit Hilfe des trainierten Modells für die Erkennung, zu erkennen.

Objekterkennungsmodell Die Modelle ermöglichen eine schnelle Erkennung. Sie werden vortrainiert und auf das Ein-Chip-System implementiert.

3.2 Stakeholder

Mit Stakeholder, auch Interessenparteien genannt, werden alle Personen, Gruppen oder Institutionen bezeichnet, die von den Aktivitäten eines Projektes direkt oder indirekt betroffen sind oder ein Interesse an diesen Aktivitäten haben. Laut des Stakeholder-Modells sollen die Beteiligten die Erwartungen und Anforderungen ihrer Stakeholder kennen und berücksichtigen. Demzufolge werden die Ziele und die Strategie des Unternehmens definiert. Hinzu wird ein Stakeholder-Konzept entwickelt und in Bezug auf das Stakeholder-Management umgesetzt. In Tabelle 3.1 werden die Stakeholder aufgelistet, die an diesem Projekt beteiligt sind.

Rolle	Kontakt	Erwartungshaltung
Auftraggeber und Produkteigentümer	Marc Hensel	Funktionsfähigkeit, Quälität des Produkts, Erfüllung der Bedürfnisse
Verantwortlicher Datenbeschaffung	Alpha Diallo	Erstellung und Aufbereitung des guten Datensatzes (Trainings-, Validierungs-, und Testdaten)
Softwareentwickler	Alpha Diallo	Softwareanforderungen und definierte Hardwareschnittstelle zum RC Fahrzeug existiert
Softwaretester	Alpha Diallo	Quälität und Funktionalität der Software prüfen und Mängel korrigieren

Tabelle 3.1: Stakeholder des Systems

3.3 Anwendungsfälle

Die Anwendungsfälle beschreiben das nach außen sichtbare Verhalten des Systems aus Sicht der Anwender. Hierbei kann der Anwender eine Person, eine Rolle oder ein anderes System sein. Als Akteur tritt dieser Nutzer mit dem System in Interaktion, um ein bestimmtes Ziel zu erreichen. Das linke Diagramm in Abbildung 3.3 gibt eine Übersicht

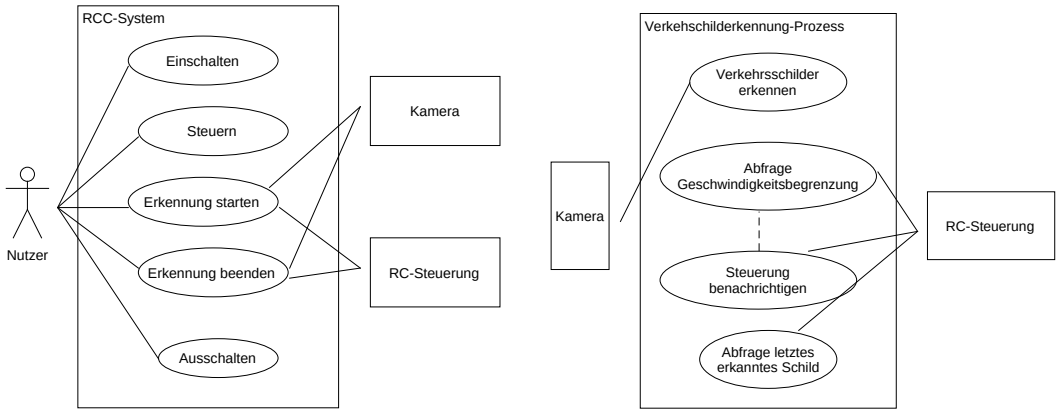


Abbildung 3.3: Anwendungsfalldiagramme

über die Funktionalität der Hardware. Der Nutzer kann das System ein-/ausschalten.

Das RC-Fahrzeug wird manuell über einen Sender durch den Nutzer gesteuert. Anschließend kann das Verkehrsschilderkennungsprogramm starten beziehungsweise stoppen. Es ist wichtig hier darauf hinzuweisen, dass das Verfahren der Verkehrsschilderkennung nur beginnt, wenn der Nutzer das Verkehrsschilderkennungsprogramm startet. Hier werden Verkehrsschilder erkannt, klassifiziert und an die Steuerung übertragen. Anhand der Benachrichtigung kann die RC-Steuerung die letzten erkannten Schilder abfragen.

3.4 Funktionale und nicht-funktionale Anforderungen

Um die Entwicklungsziele des Gesamtsystems genau zu definieren, werden in diesem Abschnitt die funktionalen und nicht-funktionalen Anforderungen beschrieben. Funktionale Anforderungen beschreiben gewünschte Funktionalitäten des Systems, während nichtfunktionalen Anforderungen die Leistung der Qualität des Systems erbringen.

3.4.1 Funktionale Anforderungen

Im Folgenden werden die funktionalen Anforderungen des Systems erläutert.

F1 Das Ein-Chip-System muss für eine Objekterkennung geeignet sein.

F2 Das System muss in der Lage sein, Kameradaten aufzunehmen.

F3 Das System muss die konkreten Verkehrsschilder erkennen. Aufgrund der großen Anzahl von Verkehrsschildern wird lediglich eine limitierte Anzahl von Verkehrsschildern betrachtet. Zu den betrachteten Verkehrsschildern gehören unter anderem Gefahrzeichen in Dreieckform, Zeichen für die Vorfahrtstraße, Vorschriftszeichen (Vorfahrtszeichen, Fahrrichtung und Geschwindigkeitsbegrenzung), Richtzeichen in blauer Rechteckform und die Richtungstafel.

F4 Das System soll eine Klassifizierung unter der Annahme, dass die Verkehrsschilder in Gruppen aufgeteilt sind, durchführen.

F5 Das System soll dazu in der Lage zu sein, aus den Vorschriftszeichen die zulässige Höchstgeschwindigkeit zu extrahieren.

F6 Der Objekterkennungsalgorithmus muss robust sein, sodass mehrere Verkehrsschilder in einem Bild erkannt werden können.

3.4.2 Nicht-funktionale Anforderungen

Die folgende Auflistung beschreibt die Nicht-funktionalen Anforderungen des Systems.

NF1 Die Software muss in der Lage sein, auf dem Ein-Chip-System zu laufen.

NF2 Das Ein-Chip-System muss über mindestens 2 GB Arbeisspeicher(RAM) verfügen, um eine effektive Erkennung zu gewährleisten. Manche Netzwerkarchitekturen, wie YOLO [49], brauchen in einigen Fällen über 1 GB Arbeitspeicherplatz für ein vortrainiertes Modell [47]. Es ist auch sinnvoll darauf hinzuweisen, dass Programme schneller ausgeführt werden, je mehr Arbeitspeicher (RAM) das System besitzt.

NF3 Für eine Echzeit-Verkehrserkennung werden mindestens 10 Frames Pro Sekunde (FPS) [6] oder höher benötigt. Das gegebene RC-Fahrzeug fährt mit einer maximalen Geschwindigkeit von 25km/h bzw. 6.94m/s. Das System soll jeden Meter ein Bild verarbeiten, sodass die Bitrate mindestens 7 FPS betragen muss.

NF4 Die Wahl der Modelle soll auf das Genauigkeit-Latenz-Verhältnis basieren. Die Modelle sollten möglichst klein sein, um die Latenz zu vermindern, aber gleichzeitig eine höhere Genauigkeit zu gewährleisten.

Im Laufe dieser Thesis werden weiterhin die Kürzel F# für die funktionalen Anforderungen und NF# für die Nicht-funktionalen Anforderungen verwendet. Dazu wird regelmäßig geprüft, ob der aktuelle Projektfortschritt zu den bereits genannten Anforderungen passt.

4 Konzept

Die Evaluierung von Objekterkennungsnetzwerken im Sinne dieser Thesis soll mithilfe von Verkehrsschildern auf einem Ein-Chip-System realisiert werden. Das RCC-System verarbeitet Kameradaten und kann so Verkehrsschilder detektieren, lokalisieren, klassifizieren und somit, wenn vorhanden, eine Geschwindigkeitsbegrenzung extrahieren. Auf Basis der formulierten Anforderungen im Abschnitt 3.4 gibt dieses Kapitel einen klaren Überblick, welche Technologie für das System verwendet wird und warum diese Technologie gewählt wurde.
Zunächst wird der Prozess der Entwicklung des Systems vorgestellt, darauffolgend gibt es einen Überblick über die Bestandteile der Hardware des RCC-System. Anhand dieses Hardware-Systems wird darüber diskutiert, welches Netzwerkmodell passend für das System ist. Dabei spielen die verfügbare Dokumentation und die angebrachte Leistung der einzelnen Netze bei der Auswahl von künstlichen neuronalen Netzen für Objekterkennung eine Rolle. Drei Netzwerkmodelle werden trainiert, getestet und validiert. Nach der Auswahl der drei Netzwerkmodelle wird entschieden, welche Software und Tools verwendet werden. Da die Anzahl von Verkehrsschildern zu groß ist, um alle existierenden Verkehrsschilder in dieser Thesis zu betrachten, werden die Art der Verkehrsschilder definiert und in Klassen aufgeteilt. Abschließend werden die Evaluationsmetriken vorgestellt, die in dieser Arbeit verwendet werden.

4.1 Übersicht

Es ist angemessen, dass das System vor der Entwicklung des Systems dargestellt werden soll. In Abbildung 4.1 wird der grobe Ablauf des gesamten Systems präsentiert.

Nach dem Aufnehmen der Kameradaten werden aus diesen Daten Verkehrsschilder lokalisiert und klassifiziert. Wenn das erkannte Schild ein Geschwindigkeitsschild ist, wird das Geschwindigkeitslimit extrahiert und an die Steuerung übertragen. Mit diesem Ablauf des Systems werden die Anforderungen F2, F3, F4 und F5 nicht verletzt.

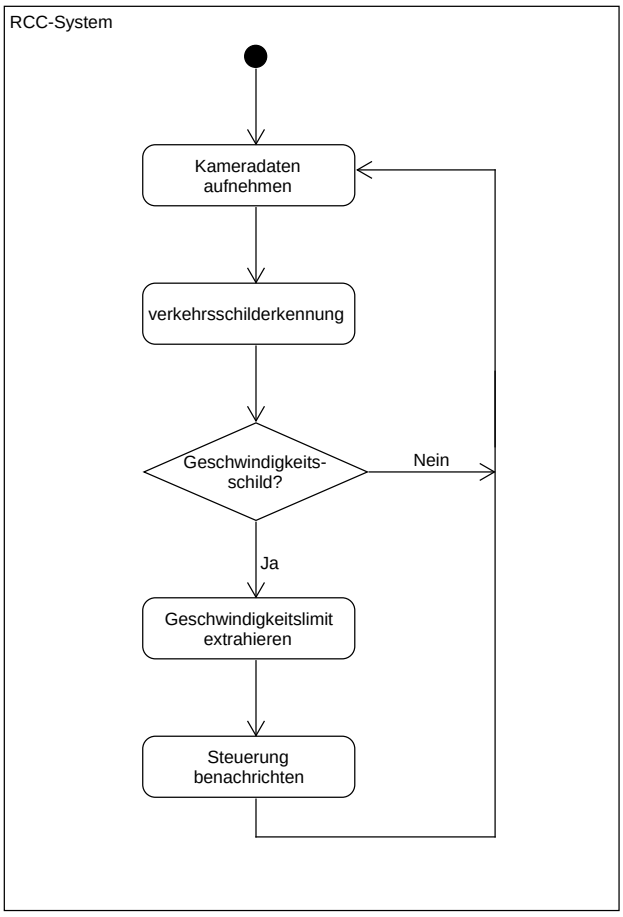


Abbildung 4.1: Aktivitätsdiagramm des Systems

4.2 Übersicht des Hardware-Systems

Für dieses Projekt wird ein Hardware-System bestehend aus drei Bestandteilen verwendet. Der erste Bestandteil ist das RC-Fahrzeug. Für die Versorgung des RC-Fahrzeugs und der weiteren Bauelemente wird eine Batterie mit einer Leistung von 3000 mAh von Herrn Hensel bereitgestellt. Das RC-Fahrzeug wird mithilfe eines Senders auf einer Frequenz von 2.4 GHz gesteuert. Hier darauf hinzuweisen, dass sowohl die Batterie als auch der Sender als Teile des RC-Fahrzeugs bezeichnet werden. Dazu wird ein RPI als Hauptrechner bzw. Ein-Chip-System verwendet und eine Kamera für die Videoübertragung benötigt.

Zunächst wird das RC-Fahrzeug vorgestellt, anschließend wird die Auswahl für den RPI und die Pi Kamera erläutert.

4.2.1 RC-Fahzeug

Ein funkferngesteuertes Auto (*engl.Radio-controlled cars "RC cars"*) oder RC-Fahrzeug ist ein Modellauto, das per Funk durch einen Sender ferngesteuert wird. Die Abbildung 1.1 stellt das RC-Fahrzeug und den Sender dar.

4.2.2 Raspberry Pi

Der Raspberry Pi ist ein kleiner Einplatinencomputer, der von der Raspberry Pi Foundation, mit Sitz im Vereinigten Königreich, entwickelt wurde. Er ist ein Rechner, der ein Ein-Chip System (SoC) von Broadcom mit einer Arm-CPU enthält. Die erste Version des Raspberry Pi, mit dem Name *Raspberry Pi Model B*, wurde im Februar 2012 veröffentlicht [56]. Seit der Veröffentlichung dieser ersten Generation wurden viele andere Version veröffentlicht. Raspberry Pi unterstützt viele Betriebssysteme, wie das Debian-basierte Betriebssystem Raspiban oder Ubuntu MATE. Er Unterstützt die Programmiersprache Python und viele Bibliotheken für neuronale Netze und Bildverarbeitung wie TensoFlow, Keras, PyTorch oder auch OpenCV. Der Raspberry Pi ist anhand all dieser Vorteile ein geeigneter Kandidat für die Wahl eines Ein-Chip-Systems, weswegen die Wahl in dieser Arbeit auf diesen fiel. Dadurch ist die Anforderung F1 erfüllt.

In Tabelle 4.1 wird ein Vergleich zwischen den verschiedenen Boards dargestellt. Als neuester und schnellster RPI ist der Pi 400 in einer kompakten Tastatur integriert. Aus Platzbedarf lässt sich der Pi 400 in dieser Arbeit nicht verwenden. Der RPI 4B ist der Vorgänger vom Pi 400, allerdings in Form einer Platine. Im Vergleich zu seinem Vorgänger ist der RPI 4B schneller in der Rechenleistung, da dieser über einen 1,5-GHz-Quad-Core-Prozessor mit 2, 4 oder auch 8 GB Arbeitsspeicher verfügt. Unter Betrachtung des Preis-Leistungs-Verhältnisses wird für diese Arbeit das Raspberry Pi 4B Modell mit 4 GB Arbeitsspeicher verwendet. Der Raspberry Pi 4B erfüllt einerseits die Anforderung NF2 und verletzt andererseits nicht die Anforderung F1.

Raspberry Pi Modell	veröffentlicht	RAM	CPU
Raspberry Pi 400	November 2020	4 GB	1.8 GHz, Broadcom BCM2711 Quad-core Cortex-A72 (ARM v8)
Raspberry Pi 4B	Juni 2019	2/4/8 GB	1.5 GHz, Broadcom BCM2711 Quad-core Cortex-A72
Raspberry Pi 3B+	März 2018	1 GB	1.4 GHz, Broadcom BCM2837B0 Quad-core Cortex-A53
Raspberry Pi 3B	Februar 2016	1 GB	1.2 GHz, Broadcom BCM2837 Quad-Core Cortex-A53

Tabelle 4.1: Modell Vergleich [44]

4.2.3 Pi Kamera

Um Videos während der Fahrt aufzunehmen, wird ein Pi Kamera V2.1 eingesetzt. Das Raspberry Pi Kameramodul 2 ersetzte im April 2016 das ursprüngliche Kameramodul. Mit einer Auflösung von 8 Megapixeln[42] kann die Pi KKamera v2 verwendet werden, um hochauflösende Videos sowie Standbilder aufzunehmen. Die Tabelle 4.2 stellt dar, welche Auflösung und FPS die Pi Kamera V2.1 unterstützt [42]. Die Kamera funktioniert mit allen Modellen von Raspberry Pi 1, 2, 3 und 4 [43]. Es gibt zahlreiche dafür gebaute Bibliotheken von Drittanbietern, einschließlich die Pi Kamera Python-Bibliothek [43]. Das bedeutet, es ergibt sich die Möglichkeit Kameradaten während der Fahrt aufzunehmen Somit erfüllt sich die Anforderung F2.

Nr.	**Auflösung**	**Frame pro Sekunde**
1	3280x2464	1/10 <= FPS <= 15
2	1920x1080	1/10 <= FPS <= 30
3	1640x1232	1/10 <= FPS <= 40
4	1640x922	1/10 <= FPS <= 40
5	1280x720	40 < FPS <= 90
6	640x480	40 < FPS <= 90

Tabelle 4.2: Sensor Modes

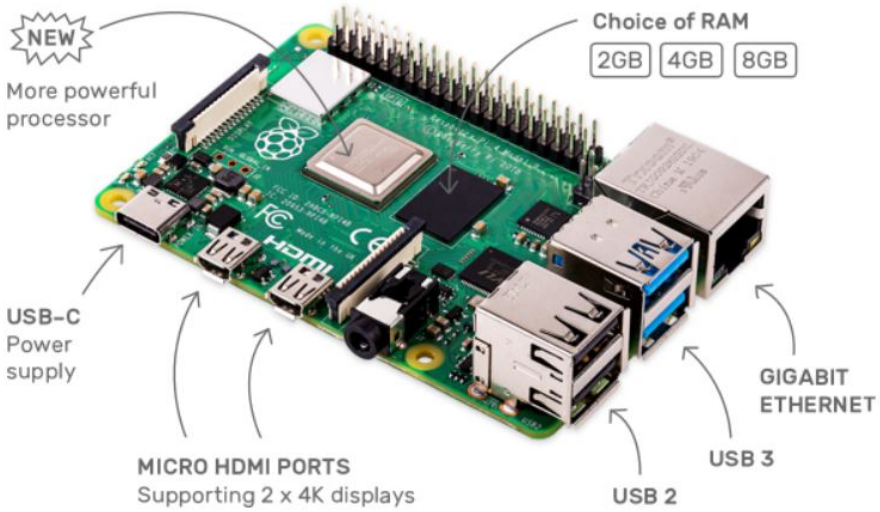


Abbildung 4.2: Raspberry Pi 4 B [44]

4.3 Auswahl des Netzwerkmodells

Eine der Komplexität von NN ist die Echtzeit-Objekterkennung und -Verfolgung, da die Hardware eine große Rolle spielt. Im Abschnitt 4.2 wurde die Hardware vorgestellt, die in dieser Arbeit verwendet wird. Die Auswahl von NN für Objekterkennungsanwendung soll unter Berücksichtigung der Leistung des RPI 4B stattfinden. Je leistungsfähiger die Hardware ist, desto schneller wird die Erkennung des Objekts durchgeführt. Aber dennoch ist die Latenz des Modells nicht nur von der Hardware des Rechners abhängig, sondern auch von der Netzwerkarchitektur. Denn einige Netzwerke haben trotz eines gleichen Hardware-Systems eine bessere Leistung als andere. In diesem Abschnitt wird entschieden, welche Modelle in dieser Arbeit zum Einsatz kommen. Dabei werden drei Modelle von drei verschiedenen Netzwerken, unter Berücksichtigung von Anforderungen F4 und NF4, ausgewählt.

Im Abschnitt 2.4.2 wurden Objekterkennungsnetzwerke vorgestellt, die auf der Github Webseite von TensorFlow Model Zoo [58] aufgelistet wurden. Zusätzlich dazu YOLO, welches nicht auf der Webseite genannt wurde. Die Netzwerke sind als Familie in Abbildung 4.4 dargestellt. Diese sind: Centernet, CNN-Familie, Efficiendent, SSD und YOLO. Um die beste Wahl zu treffen, wird eine Analyse durchgeführt, um die Auswahl der Netzwerke zu reduzieren. In der ersten Analyse werden die die CNN- und YOLO-Familie genauer betrachtet:

Abbildung 4.3: Raspberry Pi Camera Module 2 [43]

- Die CNN-Familie besteht aus RCNN, Fast RCNN, Faster RCNN und Mask-RCNN. Obwohl Fast RCNN eine Verbesserung von RCNN ist, verwenden beide Netze die selektive Suche als vorgeschlagene Methode, um die ROI zu finden. Dies ist ein langsamer und zeitaufwendiger Prozess. Aber die ROI lässt R-CNN massive Daten, Zeit, Berechnungen und Energie verbrauchen. Es erhöht die Laufzeit der RCNN-Methode. Daher benötigt RCNN fast 40 bis 50 Sekunden und Fast-RCNN fast zwei Sekunden für jedes Bild [3]. Um die Anforderung NF3 nicht zu verletzen, lassen sich RCNN und Fast RCNN schwer für diese Arbeit einsetzen.
- Obwohl die YOLO Objekterkennung von 3 verschiedenen Teams mit verschiedenem Konzept entwickelt wurde, beweist jedes Modell eine Echtzeit-Erkennungsfähigkeit. YOLOv3 ist eine Verbesserung von den Vorgängern YOLOv1 und YOLOv2, die von Redmond u.a. veröffentlicht wurden [48]. Daher werden YOLOv1 und YOLOv2 für die weitere Auswahl ausgeschlossen.

Diese erste Analyse reduziert die Anzahl der Netze für die Objekterkennung. Ein weiterer Punkt, der zu berücksichtigen ist, ist dass die Anwendung von neuronalen Netzen auf einem Ein-Chip-System keine Echtzeit-Funktionalität garantiert. Viele Netzwerke bevorzugen, durch die Fähigkeit der Parallelisierung, als Hardware eine GPU. Deswegen bieten die Entwickler eine Lite- oder Klein-Version an, um eine schnellere Geschwindig-

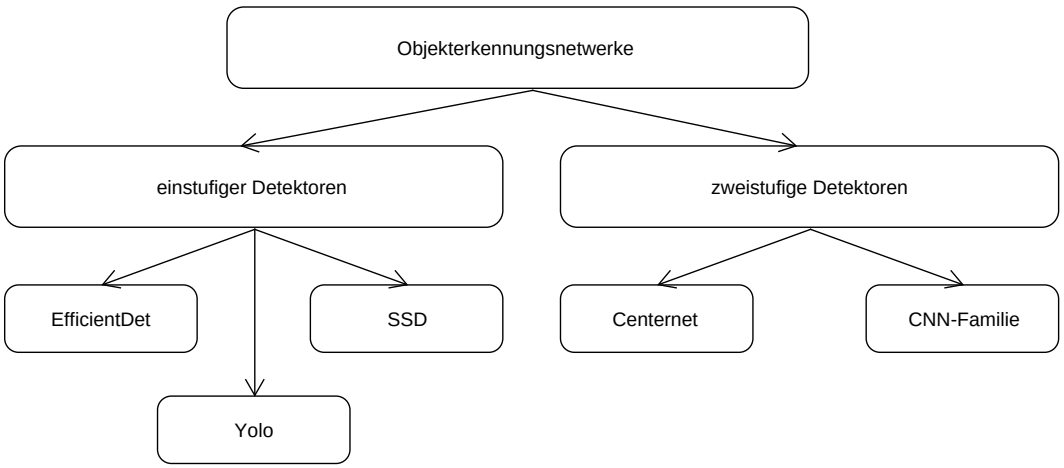


Abbildung 4.4: Arten von Objekterkennungsnetwerke laut Abschnitt 2.4.2

keit der Objekterkennung für weniger leistungsfähige Rechner zu realisieren. Unter diesen Bedingungen werden nur die Lite-Modelle der verbliebenden Netzwerke und deren Eigenschaften der ersten Analyse betrachtet. In Tabelle 4.3 sind die kleinsten Modelle jeder Familie dargestellt. Darüber hinaus werden in der nächsten Analyse Netzwerke bzw. Modelle mit einer Bounding-Box als Output betrachtet, da die Verkehrsschilder entweder in einer kreisförmigen, drei- oder viereckigen Form vorhanden sind.

Netzwerkarchitektur	Bildrate in FPS	mAP in %
CenterNet MobileNetV2 FPN 512x512	167	23.4
EfficientDet-D0	26	33.6
SSD MobileNet v2 320x320	57	20.2
Faster R-CNN ResNet50 V1 640x640	19	29.3
Mask R-CNN Inception ResNet V2 1024x1024	3	39.0
YOLOv3-Tiny	220	33.1
YOLOv4-Tiny	443	40
YOLOv5s	156	37.2

Tabelle 4.3: Überblick über Modellen von *Model Zoo* [58] und YOLO [48] [8] [27]

Die genannten Informationen entstammen für Centernet, EfficientDet, SSD, sowie Faster- und Mask RCNN von Tensorflow Model Zoo [58], für YOLOv3-Tiny von der YOLOv3

Webseite [48], für YOLOv4-Tiny von Github AlexyAb [8] und für YOLOv5s von Github ultralytics [27].
Die Tabelle 4.3 liefert zwei Maßeinheiten für jedes Modell. Die Genauigkeit mean Average Precision (mAP) iwrd in Prozent und die Bildrate in FPS angegeben. Die FPS definieren die Anzahl von verarbeiteten Bildern pro Sekunde. Um die Anforderung NF4 nicht zu verletzen, die auf dem Genauigkeit-Latenz-Verhältnis basiert, wird zuerst die mAP betrachtet. CenterNet besitzt zwar eine höhere Bildrate, aber die Genauigkeit ist gemeinsam mit dem SSD-Modell am niedrigsten. Das Faster RCNN-Modell hat im Vergleich zu den YOLO-Modellen oder des EfficientDet-Modells eine eher niedrige Genauigkeit und Bildrate. Obwohl YOLOv3 und YOLOv4 durch 2 verschiedene Teams veröffentlicht wurden, ist YOLOv4 eine Forsetzung von YOLOv3. Beide Modelle nutzen Darknet als Backbone. Im Gegensatz zu YOLOv4 besitzt das YOLOv5-Netzwerk eine PyTorch-Implementation, was eine Verbesserung bzw. Fortsetzung des YOLOv3-Netzwerks darstellt. Aufgrund dieser Ähnlichkeiten und der guten Leistung von YOLOv4-Tiny, wird YOLOv3-Tiny nicht in der weiteren Auswahl betrachtet. Damit lassen sich die drei Netze für die Arbeit auswählen. Diese sind wie folgt: EfficientDet-D0, YOLOv4-Tiny und YOLOv5s. Laut den Informationen aus Tabelle 4.3 erfüllen die drei Modelle ebenfalls die Anforderung NF3.

4.4 Software

Nachdem die Auswahl der Netzwerkmodelle und der Komponenten des Hardwaresystems in den Abschnitten 4.2 und 4.3 vorgestellt wurden, wird in diesem Abschnitt festgelegt, welche Software und Bibliotheken verwendet werden. Zunächst wird im Abschnitt 2.3.2 eine programmierte Sprache ausgewählt. Anhand der Programmiersprache wird die Entwickungsumgebung festgelegt. Darüber hinaus werden die Schnittstellen für die Modelle definiert. Abschließend wird das Konzept über das Extrahieren der Gewschwindigkeitsbegrenzung der Gewschwindigkeitsschilder vorgestellt.

4.4.1 Programmiersprache

In Abschnitt 2.3.2 wurde C++, MATLAB und Python vorgestellt. Diese Programmiersprachen unterstützten ML-Aufgaben. Allerdings wird die ganze Entwicklung auf Python 3 basieren. Die Bibilotheken in Teilabschnitt 2.3.1 verfügen über Python-Schnittstellen. Python wird mit dem Booten von Raspbios RPI 4B mit installiert, wodurch sich die Zeit

für die Installation entfällt. Einer der Hauptvorteile der Verwendung von Python ist die Möglichkeit, direkt mit dem Code zu interagieren, indem ein Terminal oder andere Tools wie das Jupyter Notebook verwendet werden. Machine Learning und Deep Learning sind grundsätzlich iterative Prozesse, bei denen die Daten die Analyse vorantreiben. Für diese Prozesse ist es wichtig, über Tools zu verfügen, die eine schnelle Iteration und einfache Interaktion ermöglichen [36]. Im Vergleich zu Python hat C++ eine komplexe Syntax, da die Programmiersprache C++ nah an der Maschinensprache liegt. Einzig Matlab ist auf Lizenzen basiert, wodurch hat MATLAB eine kleinere Community im Vergleich zu Python besitzt. Dies hat den Nachteil, dass sich wenig Lösungansätze für gewisse Problem finden lassen. TensorFlow Zoo YOLOv4 und YOLOv5 haben bereits Programme in ihren Github für das Training und die Evaluation der Modelle bereitgestellt, welche auf Python basieren. Diese genannten Gründen stellen Python als den beste Kandidaten für diese Arbeit dar.

4.4.2 Entwicklungsumgebung

Jupyter Notebook wird für diese Arbeit verwendet. Jupyter Notebook ist eine browserbasierte Anwendung. Sie unterstützt Python und lässt sich leicht auf RPI installieren. Die Ertellung bzw. Bearbeitung von Programmen mit Jupyter Notebook funktioniert dabei über die webbasierte Client-Anwendung, die sich mit handelsüblichen Browsern starten lässt. Voraussetzung ist, dass auf dem System auch der Jupyter-Notebook-Server installiert ist und ausgeführt wird. Die erstellten Jupyter-Dokumente lassen sich u. a. als HTML-, PDF-, Markdown- oder Python-Dokumente exportieren oder alternativ per E-Mail, Dropbox, GitHub oder dem hauseigenen Jupyter Notebook Viewer mit anderen Nutzern teilen.

4.4.3 Schnittstelle für Objekterkennungsmodell

Viele Objekerkennungsnetzwerke lassen sich unter der Anwendung verschiedener Schnittstellen durchführen. Von den ausgewählten Modellen für diese Arbeit, verwendet YOLOv4 als Standard-Schnittstelle für die Implementierung des Netzwerkes Darknet und YOLOv5 verwendet PyTorch. Dennoch lassen sich für beide Netzwerke andere Schnittstellen wie beispielweise TensorFlow object detection oder PyTorch für YOLOv4 verwenden. EfficientDet lässt sich von TensorFlow object detection (TF2x 2x für die aktuell Version) leicht unterstützen. Die API hat eine gute Dokumentation und TF2x untersützt

und erleichtert die Vorgehensweise für das Training eines Modells. Aufgrund dieser Gründe werden YOLOv4 und YOLOv5 ihre Standardschnittstellen und EfficientDet TF2x verwenden.

4.4.4 Extraktion von der maximalen Geschwindigkeit

In diesem Teilabschnitt wird erläutert, welche Technologien angewendet werden, um die maximale Geschwindigkeit aus Geschwindigkeitsschildern zu extrahieren. Unter Betrachtung des Aktivitätsdiagramms 4.1 werden nach einer erfolgreichen Erkennung mittels EfficientDet, YOLOv4-Tiny oder YOLOv5 die maximale Geschwindigkeit eines Geschwindigkeitsschildes extrahiert. Die maximale oder vorgeschriebene zulässige Geschwindigkeit ist nicht nur durch Schilder mit Zahlen in der deutschen Verkehrsordnung zu ermitteln. Schilder wie *Stop* weisen eine Verminderung der Geschwindigkeit. Das lässt sich in dieser Arbeit aus zeitlichen Gründen schwierig umsetzen. Deshalb wird hier die Extraktion von der maximalen Geschwindigkeit aus Schildern mit Zahlen betrachtet. Mit dieser Annahme lässt sich ein OCR Textleser anwenden. Unter vielen OCR-Modellen wird EasyOCR für diesen Teil der Arbeit angesetzt. Somit erfüllt sich die Anforderung F6.
Optical Character Recognition (OCR) ist eigentlich ein kompletter Prozess, bei dem die Bilder/Dokumente, die in einer digitalen Welt vorhanden sind, verarbeitet und aus dem Text als normaler bearbeitbarer Text verarbeitet werden. OCR ist eine Technologie, mit der Sie verschiedene Arten von Dokumenten, z. B. gescannte Papierdokumente, PDF-Dateien oder von einer Digitalkamera aufgenommene Bilder, in bearbeitbare und durchsuchbare Daten konvertieren können. EasyOCR ist eigentlich ein Python-Paket, das PyTorch als backend funktioniert. EasyOCR erkennt den Text aus Bildern mit PyTorch als Backend. EasyOCR unterstützt viele Sprachen für Erkennungszwecke und wird von der Firma Jaided AI Company entwickelt. Darüber hinaus wird auch für die Extraktion OpenCV verwendet. Wie bereits im Abschnitt 2.3.1 erwähnt, lässt sich OpenCV sehr leicht für Bildverarbeitungszwecke nutzen.

4.5 Datensatzaufbereitung

In Teilabschnitt 2.4.3 wurden einige Open-Source Bilddatenbanken für Objekerkennungzwecke vorgestellt. Diese sind: ImageNet, MS COCO, ExDark, LISA Traffic Sign Detection Dataset, Open Image und Pascal VOC-Datensatz. Allerdings liefert keine von diesem

Bilddatenbanken konkrete Datensätze um die Anforderung F3 zu erfüllen, obwohl die Datensätze LISA und Open Images über Verkehrsschilder als Kategorie verfügen. Beide bieten einen amerikanischen Verkehrersschilder-Datensatz. Nach einigen Besprechung mit Herrn Hensel wurde entschieden, dass mit Hilfe des RC-Fahrzeugs und der Pi-Kamera ein eigener Datensatz für diese Arbeit erzeugt wird.
Nach der Vorstelleung der Datensatzerzeugung, werden die vorhandene Daten mithilfe des Abschnitts 2.6 in Gruppen aufgeteilt. Dazu werden die Annotierungssofwares vorgestellt und eine davon ausgewählt. Abschließend wird der Datensatz in dem Format Trainings-, Test- und Valiedierungsdaten erläutert.

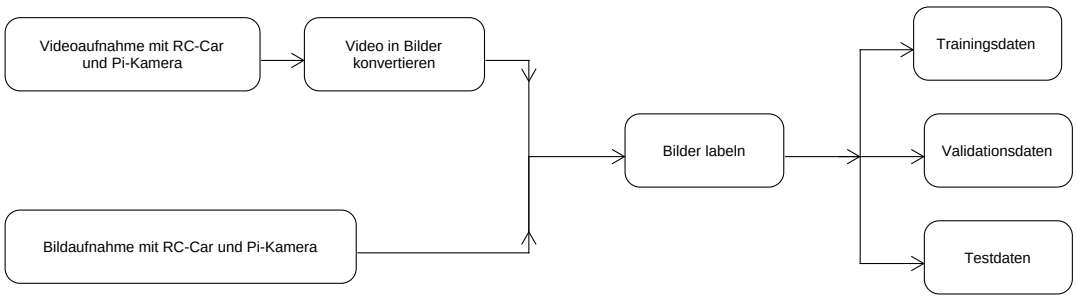


Abbildung 4.5: Workflow des Datensatzes

4.5.1 Datensatzerzeugung

Die Datesatzerzeugung wird mittels der Pi Kamera durchgeführt, die mit dem RPI 4B verbunden ist und sich auf einem RC-Fahrzeug befindet. Für die Aufnahme wird ein Raspberry -Pi 3B verwendet. Die Datensatzerzeugung wird in zwei Szenarien durchgeführt. Zuerst werden Bilder mit Hilfe des RC-Fahrzeugs in einem Abstand von einer halben Sekunde während der Fahrt aufgenommen. Dieses erste Szenario wird mehrfach repetiert, aber jedoch mit Änderung der Verkehrschildpositionierung und der Geschwindigkeit des Fahrzeugs. Die Bilder werden in verschiedenen Auflösungen aufgenommen. Das zweite Szenario besteht darin, ein Video aufzunehmen. Die Videoaufnahme wird ein paar mal, mit verschiedenen Szenen und Geschwindigkeiten des Fahrzeugs wiederholt. Diese neiden Szenarien sollten sowohl einen guten und zahlreichenden Datensatz, als auch eine gute Genauigkeit beim Testen mit dem RPI 4B gewährleisten.

Abbildung 4.6: Aufgenommene Bilder mit Hilfe der Pi-Kamera

4.5.2 Datensatzskaterosierung

Es gibt drei Gruppen der Verkehrszeichen: Gefahrzeichen, Vorschriftszeichen und Richtzeichen. Die Gefahrzeichen weisen den Verkehrsteilnehmer daraufhin, die Aufmerksamkeit zu erhöhen. Unter diese Art von Verkehrszeichen zählen Schilder in der dreieckigen Form mit rotem Rand wie beispielsweise die Kurve (rechts oder links) für die Warnung, die Verringerung der Geschwindigkeit oder Bremsbereitschaft. Die Vorschriftszeichen fassen Gebote und Verbote. Die Geschwindigkeitsbegrenzungen, die Fahrtrichtungen sowie die Verbote für Verhaltensweisen zählen zu dieser Kategorie. Unter der Kategorie Richtzeichen gehören die Schilder wie die Vorfahrt (an der nächsten Kreuzung), die Vorfahrtsstraße, Parken oder auch Autobahn und enthalten sowohl Gebote als auch Verbote. Diese Kategorisierung lässt sich schwer für dieses Projekt umzusetzen, da es sehr viele Klassen gibt und eine große Menge an Ressourcen und Zeit benötigt wird. Ein Weiterer Grund ist, dass es für dieses Projekt nicht genug Daten für die jeweiligen Klassen für das Training gibt. Darum wird eine neue Klassifizierung eingeführt, die sowohl auf Basis der Geometriedarstellung als auch die Art und Weise des Schildes referenziert. Diese neue Klassifizierung basiert auf der Art der vorhandenen Verkehrsschilder im Raum 13.60 im Gebäude Berliner 7. Daraus sind sechs neue Klassen entstanden: Gefahr, Vorfahrt, Vorschrift, Ende Vorschrift, Richtzeichen, und Einrichtung. Die Abbildung 4.7 gibt einen Überblick über die neue Klassifizierung.

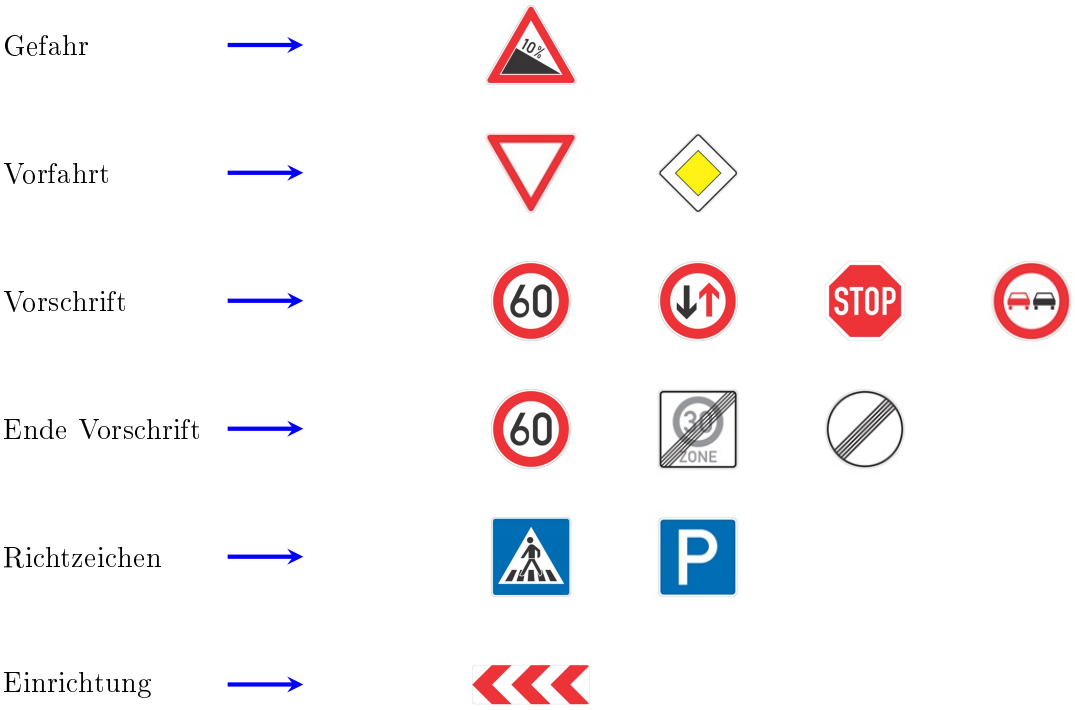


Abbildung 4.7: Angewendete Klassen von Verkehrschilder [2]

4.5.3 Annotierung des Datensatzes

Um ein robustes Modell zu trainieren, sollten die Bilder so vielfältig sein wie möglich. Sie sollten unterschiedliche Hintergründe und unterschiedliche Lichtverhältnisse haben. Die Bilder müssen mit der entsprechenden Bounding-Box Klasse annotiert werden. Es gibt zahlreichende Annotierungstools, die sich von der Anwendung oder vom Annotierungsformat unterscheiden. Alle weitere Informationen in diesem Abschnitt stammen von [25] Hier werden einige Annotierungstools aus dem Teilabschitt 2.4.1 erläutert:

- labelme ist ein auf Python basierendes Open-Source-Annotationstool zur polygonalen Annotation von Bildern, das für die manuelle Annotation von Bildern zur Objekterkennung, Segmentierung und Klassifizierung verwendet werden könnte. Labelme ermöglicht die Erstellung von Polygonen, Rechtecken, Kreisen, Linien, Punkten oder Linienstreifen. Mit diesem Tool können die Label jedoch nur als JSON-Dateien direkt aus der App heraus gespeichert werden. Für andere Formate, kann ein Python-Skript aus dem labelme-Repository verwendet werden, um die

Annotationen in PASCAL VOC zu konvertieren. Das Tool ist zuverlässig mit einer einfachen Funktionalität für die manuelle Bildannotation, aber jedoch eignet es sich für eine kleine Datensatzannotierung.

- CVAT ist ein Open-Source-Werkzeug für Bilder und Videos, für Aufgaben wie Objekterkennung, Segmentierung und Klassifizierung. Es ist möglich, die Web-Version dieses Tools online zu nutzen. Sie können gemeinsam als Team an der Annotation von Bildern arbeiten und die Arbeit zwischen den Benutzern aufteilen. Es gibt auch eine großartige Option, die es Ihnen ermöglicht, vortrainierte Modelle zu verwenden, um Ihre Daten automatisch zu labeln. Dies vereinfacht den Prozess für die gängigsten Klassen (z. B. die in COCO enthaltenen), wenn vorhandene verfügbare Modelle im CVAT-Dashboard verwendet werden. Alternativ ist es ebenfalls möglich, Ihre eigenen vortrainierten Modelle zu nutzen. CVAT hat von den bisher betrachteten Tools den größten Funktionsumfang. Insbesondere erlaubt es, Labels in etwa 15 verschiedenen Formaten zu speichern.

- hasty.ai ist im Gegensatz zu allen oben genannten Tools kein kostenloser Open-Source-Dienst. Es ist aufgrund der sogenannten KI-Assistenten für die Objekterkennung und Segmentierung sehr komfortabel für das Labeling von Daten. Der Annotationsprozess könnte mit dem automatischen Assistenten deutlich beschleunigt werden, denn ein Assistentenmodell wird während der Annotation trainiert. Das heißt, je mehr Bilder annotiert werden, desto genauer arbeitet der Assistent. hasty.ai bietet Die Testversion mit 3000 Credits, was ausreicht, um automatisch vorgeschlagene Labels von etwa 3000 Objekten für eine Objekterkennungsaufgabe zu generieren. Das Tool erlaubt es Ihnen, Daten in den Formaten COCO oder Pascal VOC zu exportieren. Ein anderer Vorteil von hasty.ai ist die Teamarbeit und die Möglichkeit Rollen in den Projekteinstellungen zuzuweisen.

- labelImg ist ein weit verbreitetes Open-Source-Werkzeug für grafische Annotationen. Es ist in Python geschrieben und verwendet QT für seine grafische Oberfäche. Es ist nur für die Lokalisierung oder Erkennung von Objekten geeignet und kann nur rechteckige Boxen um die betrachteten Objekte erstellen. Die Anwendung vom Tool ist nur auf die Erstellung von Bounding Boxes ausgerichtet, was das Tool so weit wie möglich vereinfacht. Für diese Aufgabe verfügt labelImg über alle notwendigen Funktionen und komfortable Tastaturkürzel. Ein weiterer Vorteil ist, dass Sie Annotationen in 3 gängigen Annotationsformaten speichern/laden können: PASCAL VOC, YOLO und CreateML.

Abbildung 4.8: LabelImg Tool

Nach der Betrachtung der Vor-und Nachteile der einzelnen Tools, wird LabelImg für seine Vereinfachte Anwendung als Annotierungstool gewählt .

4.5.4 Aufteilung des Datensatzes

Hier wird der Datensatz in Trainings, Test- und Valiedierungsbilder aufgeteillt. Die Aufteilung besteht zu 90% aus den Trainings- und Validierungsdaten und zu 10% aus den Testdaten. Die Trainings- und Validierungsdaten werden verwendet, um die Parameter des NNs anzupassen und das Netz besser zu Trainieren. Dabei werden unterschiedliche prozentuale Kombinationen der Trainigs- und Validierungsdaten verwendet, um den Einfluss auf das Trainingsergebnis zu beobachten. Nach dem Training und der Valiedierung werden die Daten des trainierten Modells mit den Testdaten überprüft, um die Effizienz des Modells zu bewerten.

4.6 Evaluationsmetriken

Hier werden die Evaluationsmetriken vorgestellt, die für diese Arbeit verwendet werden.

4.6.1 Non Maximum Suppression

Die Non Maximum Suppression (NMS) als Nachbearbeitungsschritt für die Objekterkennung wird hauptsächlich verwendet, um redundante Begrenzungsrahmen im Objekt zu entfernen und spielt in vielen Detektoren eine wichtige Rolle. Seine Positionierungsgenauigkeit hängt hauptsächlich von der Bounding Box mit der höchsten Punktzahl ab. Diese Strategie zielt darauf ab, falsch positive (*engl.* false positiv) Ergebnis zu eliminieren. Mit anderen Worte ist das Ziel des NMS Bounding Boxes zu kombinieren, die zu demselben Objeckt gehören, um idealerweise eine Bounding Box je Objekt anzupassen. Das Prizip ist wie folgendes:

- Bounding boxes verwerfen, wennn Confidence Level unterhalb der Schwelle,
- Wähle die Bounding Box mit dem höchsten Confidence Level,
- Berechne Überlappung zu allen verbleibenden Boxes mit demselbem Label,
- Verwerfe alle Bounding Boxes mit einer Überlappung unterhalb einer Schwelle,
- Kombiniere die verbleibenden Bounding Boxes des Labels zu einer, die alle beinhaltet.

4.6.2 Evaluationsmetriken

Die Evaluationsmetriken spielen eine entscheidende Rolle in NN um die Leistung von NN zu bewerten [51]. EfficientDet-, YOLO-v4- und YOLOv5-Netzwerke erzeugen beim Trainieren und der Auswertung ihre Evaluationsmetriken. Das erleichtert wesentlich die Arbeit. Bevor die Arten von Evaluationsmetriken untersucht werden, werden einige Konzepte erwähnt und kurz erklärt. Die grundlegendsten Evulationsmetriken sind die wie folgt definierten:

- True Positive (TP) - die Anzahl der von der Maschine richtig klassifizierten Objekte, die vom maschinellen Lernsystem als positiv gehalten wird;
- True Negative (TN) - die Anzahl der von der Maschine richtig klassifizierten Objekte, die vom maschinellen Lernsystem als negativ gehalten wird;
- False Positive (FP) - die Anzahl der Objekte aus der negativen Klasse, die vom maschinellen Lernsystem für positiv gehalten wird;

- False Negative (FN) - die Anzahl der Objekte aus dem Negativen Klasse, die vom maschinellen Lernsystem für negativ gehalten wird.

	Vorhersage negativ	Vorhersage positiv
tatsächlich negativ	true negative TN	false positive FP
tatsächlich positiv	false negative FN	true positive TP

Tabelle 4.4: . Confusion matrix [19]

Precision

Die Präzision (*engl.* Precision) ist die Fähigkeit eines Modells, nur relevante Objekte zu identifizieren. Es ist der Prozentsatz der korrekten positiven Vorhersagen und wird wie in der Gleichung 4.1 berechnet.

$$Precision = \frac{TP}{TP + FP}. \tag{4.1}$$

Recall

Recall ist die Fähigkeit eines Modells, alle relevanten Fälle, auch Ground-Truth-Bounding-Boxen genannt, zu finden. Es ist der Prozentsatz der korrekten positiven Vorhersagen unter allen gegebenen Grundwahrheiten. Die Gleichung 4.2 stellt dar, wie Recall berechnet wird.

$$Recall = \frac{TP}{TP + FN}. \tag{4.2}$$

Intersection over union (IOU)

Intersection over union (IOU) ist eine Auswertungsmetrik, die verwendet wird, um die Genauigkeit eines Objekterkennungsmodells für einen bestimmten Datensatz zu messen [39]. Im Objekterkennungsbereich misst der IOU den überlappenden Bereich zwischen

der vorhergesagten Bounding Box und der Ground-Truth-Bounding-Box geteilt durch den Bereich der Vereinigung zwischen ihnen. Da heißt:

$$IOU = \frac{FlaecheB_p \cap B_g}{FlaecheB_p \cup B_g} = \frac{Intersection}{Union}. \tag{4.3}$$

Durch den Vergleich des IOU mit einem bestimmten Schwellenwert können wir eine Erkennung als korrekt oder falsch klassifizieren. Wenn $IOU >= t$ wird die Erkennung als korrekt angesehen. Wenn $IOU < t$ wird die Erkennung als falsch betrachtet.

Average Precision AP

Die durchschnittliche Präzision (AP) ist eine Möglichkeit, die Präzisions-Recall-Kurve in einem einzigen Wert zusammenzufassen, der den Durchschnitt aller Präzisionen darstellt. Der AP wird gemäß der nächsten Gleichung berechnet. Mithilfe einer Schleife, die alle Genauigkeiten durchläuft, wird die Differenz zwischen dem aktuellen und dem nächsten Rückruf berechnet und dann mit der aktuellen Genauigkeit multipliziert. Mit anderen Worten, der AP ist die gewichtete Summe der Genauigkeiten bei jedem Schwellenwert, wobei die Gewichtung die Erhöhung des Abrufs ist.

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] \cdot Precisions(k) \tag{4.4}$$

Wobei $Recalls(k) = 0$, $Precisions(k) = 1$ und $n =$ Anzahl der Schwellenwerte.

Mean Average Precision (mAP) Normalerweise werden die Objekterkennungsmodelle mit unterschiedlichen IOU-Schwellenwerten bewertet, wobei jeder Schwellenwert andere Vorhersagen als die anderen Schwellenwerte liefern kann. MAP (Mean Average Precision) ist der Durchschnitt von AP.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4.5}$$

Loss Funktion

Die Loss Funktion ist eine Technik, um die schlechte Leistung des aktuellen Modells zu messen, welches das aktuelle Einkommen und die geschätzte Leistung bestimmt. Die allgemeine Verlustfunktion ist eine gewichtete Summation des Lokalisierungsverlustes (Lloc) mit dem Vertrauensverlust (Lconf):

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + aL_{loc}(x, l, g)) \tag{4.6}$$

In Gleichung (4.6) repräsentiert N die Anzahl der positiven entsprechenden Standardbegrenzungsboxen, wobei der Parameter a für das Gewicht des Lokalisierungsverlustes steht. Der Lokalisierungsverlust ist ein glatter L1-Verlust zwischen den erwarteten Bounding-Box- und Ground-Truth-Box-Parametern. Es ist die Fehlanpassung zwischen der erwarteten Boundary-Box und der Ground-Truth-Box. Der Lokalisierungsverlust wird für positive Kästchen berechnet und negative Übereinstimmungen können vernachlässigt werden. Die Vorhersagen aus den positiven Matches entscheiden sich dafür, der Ground Truth näher zu kommen. Der Vertrauensverlust ist der Verlust beim Erstellen einer Klassenerwartung. Der Verlust wird gemäß dem Konfidenzwert der Matching-Klasse für jede positive Match-Erwartung bestraft. Bei negativen Übereinstimmungserwartungen wird der Verlust gemäß dem Konfidenzwert der Klasse bestraft, bei der kein Objekt entdeckt wird.

4.7 Hyperparameter

Hyperparameter sind Parameter, die gesetzt werden, bevor der Lernprozess beginnt. Diese Parameter sind einstellbar und können direkt darauf auswirken, wie gut ein Modell trainiert. Hyperparameter können einen direkten Einfluss auf das Training eines neuronalen Netzes haben, um eine maximale Leistung zu erzielen. Die verwendeten Hyperparameter in dieser Arbeit sind folgende:

Batch size oder Stapelgröße ist die Anzahl der Proben, in dem Fall dieser Arbeit Bilder, die das neuronale Netzwerk normalerweise gleichzeitig durchlaufen. Diese Anzahl hängt vom Speicherplatz in der CPU oder GPU ab.

Epoche Die Anzahl der Epochen entspricht der Häufigkeit, mit der der Algorithmus den gesamten Datensatz sieht. Also ist jedes Mal, wenn der Algorithmus alle Stichproben im Datensatz gesehen hat, eine Epoche abgeschlossen.

Steps ist die Häufigkeit, mit der der Algorithmus den gesamten Datensatz sieht. Wenn der Algorithmus alle Trainingsdaten im Datensatz gesehen hat, ist eine Epoche abgeschlossen.

4.8 Zusammenfassung

In Tabelle 4.8 ist eine Zusammenfassung der Informationen, welche in diesem Kapitel erörtert wurden. Diese Informationen werden im Kapitel 5 verwendet.

Netzwek	EfficientDet	YoloV4	YoloV5
Modell	EfficientDet-D0	YoloV4-Tiny	Yolov5s
Schnittstelle	Tensorflow	Darknet	Pytorch
Config File	pipeline.config	yolov4-tiny.cfg	yolov5s.yaml
Auflösung	512x512	416x416	415x415
Hyperparameters	Steps	Batch Size	Batchs Size, Epochs

Tabelle 4.5: Angewandte Modellen für Objetkerkennung in dieser Arbeit

5 Umsetzung

Die konkrete Überlegung im Kapitel 4 hat als Konsequenz, dass 3 Netzwerke für das Hardware-Systeme ausgewählt wurden. Die 3 Netzwerke, welche EfficientDet, Yolo v4 Lite, and Yolo v5 sind, werden mit demselben Datensatz trainiert, validiert und auf dem RCC-System getestet. Das Training und der Validation-Prozess wird für alle Modelle auf einem Desktop-PC an der HAW Hamburg mit einem AMD Ryzen 9 3950X 16-Core Prozessor und einer GeForce RTX 2080 Ti als GPU durchgeführt. Dies soll die Trainingszeit deutlich verkleinern. In diesem Kapitel wird die Datensatzaufteilung vorgestellt. Anhand dieses Datensatzes werdend die 3 Netze trainiert, validiert und getestet sowie in Modell-Form exportiert. Anschließend wird die Extraktion der Geschwindigkeit entwickelt, mit den Modellen kombiniert und auf dem RPI simuliert.

5.1 Datensatz

Im Kapitel 4.5 wurde die Datensatzbeschaffung erläutert. Der Datensatz wird in Trainings-, Validierungs- und Testdaten aufgeteilt. Die Trainings- und Validierungsdaten werden für das Training angewendet. Die Abbildung 5.1 gibt einen Überblick über den Anteil jeder Klasse. Laut der YOLOv4-Entwickler soll die Anzahl der Bilder des Datensatzes für ein gutes Ergebnis pro Klasse mindestens 1500 Bilder betragen [8]. Die Bedingung ist leider nicht erfüllt und die Auswirkung wird in der Auswertung ausgewertet. Die Anzahl von Bildern für den Datensatz in dieser Thesis beträgt 1088 aufgenommene und gelabelte Bilder. Die Aufteilung der Bilder erfolgt per Zufall. 10% der Bilder werden jedoch für die Testdaten verwendet, welche für den Test auf dem RPI benutzt werden. Daraus ergeben sich 109 Bilder für den Testdatensatz. Um ein besseres Trainingsergebnis zu erzielen wird aus den Übrigen 979 Bilder eine Kombination von verschiedenen Verhältnissen verwendet, wie beispielsweise 80/20 oder 90/10 oder sogar 70/30.

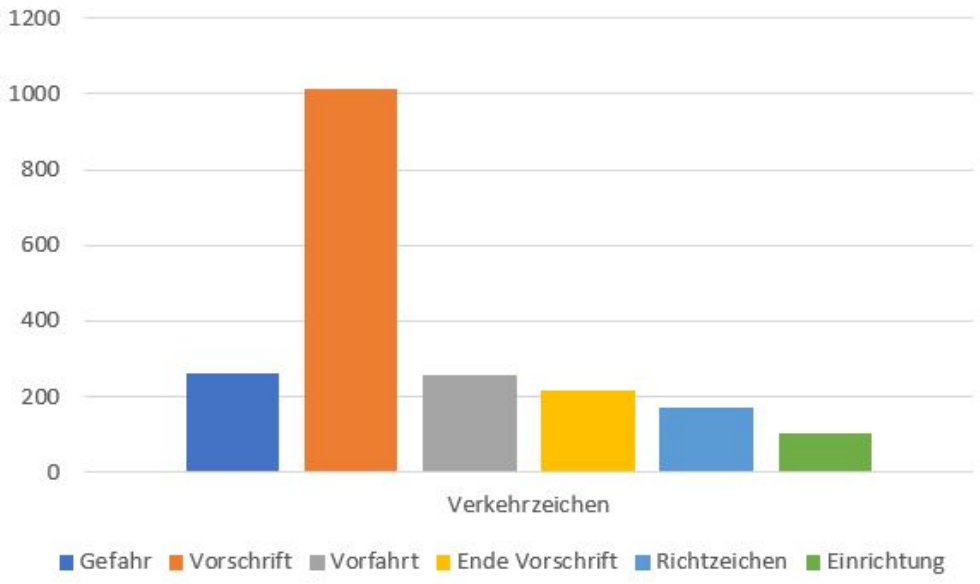


Abbildung 5.1: Anzahl jeder klasse im Datensatz

5.2 Training der Modelle

In diesem Abschnitt wird der Trainingsprozess erläutert. Dieser Prozess wird für jedes Modell selbst, in der Reihenfolge wie es trainierte wurde, erklärt. Außerdem wird eine Hyperparameter-Optimierung durchgeführt, wie es bereits in Kapitel Konzept, Abschnitt 4.7 erklärt wurde.

5.2.1 Yolov5s

Um das Netz zu trainieren, wird das Yolov5-Repository von Github geklont. Alle nötigen Installationsanforderungen, einschließlich PyTorch>=1.7. Modelle und Datensätze werden automatisch von der neuesten YOLOv5-Version heruntergeladen. Alle YOLOv5-Modelle müssen mit gelabelt Daten trainiert werden, um Klassen von Objekten in diesen Daten zu erlernen. Diese können im YOlO-Format sein. Zur Verbesserung des Trainingsergebnisses werden die Hyperparameter Batch und Epochen optimiert. Darüber hinaus wird mit unterschiedlichen Verhältnissen oder Ratio der Anteile von Trainings- und Validierungsdaten herumprobiert, um ein besseres Trainingsergebnis zu erzielen.

Ratio Bilder	batch	epoch	Genauigkeit (mAP)	Trainingszeit
90/10	8	100	96.2%	0.275h
90/10	8	300	91.6%	0.586h
80/20	16	100	87.8%	0.233h
80/20	16	300	88.4%	0.538h
90/10	16	300	95.6%	0.480h
90/10	16	100	96.5%	0.248h
80/20	32	100	87.1%	0.221h
80/20	32	300	87.0%	0.576h
90/10	32	100	95.4%	0.224h
90/10	32	300	94.0%	0.668 h

Tabelle 5.1: Trainingsergebnisse von YOLOv5s

Die Tabelle 5.1 liefert die Trainingsergebnisse mit verschiedener Anzahl von Batch und Epochen sowie einer prozentuale Ratiokombination. Die Anzahl der Batches soll in der Regel durch 4 teilbar sein, wobei die meist angewendeten Anzahlen 8, 16, 32, 64, 12, 126 und 256 sind. Aufgrund des Speicherbedarfes sind Batch = 32 die maximale einstellbare Anzahl von Batches für die verwendete GPU. Angemerkt ist, dass die beste Genauigkeit bei mAP mit den Werten Batch = 16, Epoche = 100 und Ratio = 90/10 erzielt wird.

5.2.2 YOLOv4-Tiny

Die Vorbereitung für das Trainieren von YOLOv4-Tiny erfordert Zeit. Zuerst wird das Netz von Github geklont. OpenCV und Cmake sind erforderliche Anforderungen für das Trainieren von YOLOv4. Dieses Netzwerkmodell verwendet wie alle YOLO-Modelle die YOLO-Txt-Formate für die Annotierung. Darüber hinaus ist im Allgemeinen das Kompilieren von Darknet für YOLO-v4 erforderlich. Weitere Konfigurationen werden anhand der Konfigurationsdatei *yolov4-tiny-custom.cfg* eingestellt, wie beispielsweise die Anzahl von Batches, Klassen oder Filter. Darüber hinaus ist für das Trainieren von YOLOv4-Tiny erforderlich, dass die Max-Batches 2000 * die Anzahl von Klassen nicht übersteigen soll [8]. Die Tabelle 5.2 liefert die Trainingserbnisse mit eine Variation der Anzahl von Batch und der Ratio. Das beste Ergebnis liegt bei 91,78% mit Batch = 64 und Ratio = 90/10.

Ratio	Batch	Genauigkeit (mAP)	Trainingszeit
20/80	16/16	87.21%	0,33h
90/10	16/16	91.11%	0,51h
90/10	32/16	91.11%	0,53h
20/80	32/16	87,91%	0,55h
30/70	64/16	81.65%	0.97h
80/20	64/16	87.53%	0.96h
90/10	64/16	91,78%	0,93h

Tabelle 5.2: Trainingsergebnisse des YOLOv4-Tiny Modells

5.2.3 EfficientDet-D0

In diesem Abschnitt wird das Training des Modells EfficientDet-D0 unter TensorFlow 2x Object detection (TF2x) erläutert. Auch hier wird das TensorFlow model Repository von Github geklont, also eine Reihe verschiedener Implementierungen von Modellen und Modellierungslösungen für TensorFlow-Benutzer [58]. Anhand dieses Repository wird die Object Detection API installiert. Der Datensatz wird zuerst in xml annotiert, dann in das CSV-Format umgewandelt und abschließend wird der Datensatz mithilfe des CSV-Files in das TFRecord-Format konvertiert. Aufgrund eines höheren Speicherplatzbedarfs lässt sich EfficientDet-D0 unter der angewendeten GPU nur mit einem Batch = 4 oder 8 trainieren. Um mit einer höheren Anzahl von Batches zu trainieren, wird das Training unter der CPU durchgeführt. Wie in Tabelle 5.3 dargestellt, sind das Trainieren von EfficientDet-D0 sehr zeitaufwändig. Die maximale Anzahl von Batches unter der CPU ist 32. Für Batches größer als 32 ist der Hauptspeicher nicht ausreichend. Das beste Ergebnis liegt bei einer Genauigkeit von 75.84% für 32 Batches und einer Ration von 90/10.

Modell	Ratio Bilder	Batch	Accuracy (mAP)	Trainingszeit (std)
Efficident-D0	20/80	8	54.57%	134h
Efficident-D0	90/10	8	58.19%	1.42h
Efficident-D0	90/10	16	71.32%	48,61h
Efficident-D0	90/10	32	75.84%	49,58h

Tabelle 5.3: Trainingserbnisse von EfficientDet-D0

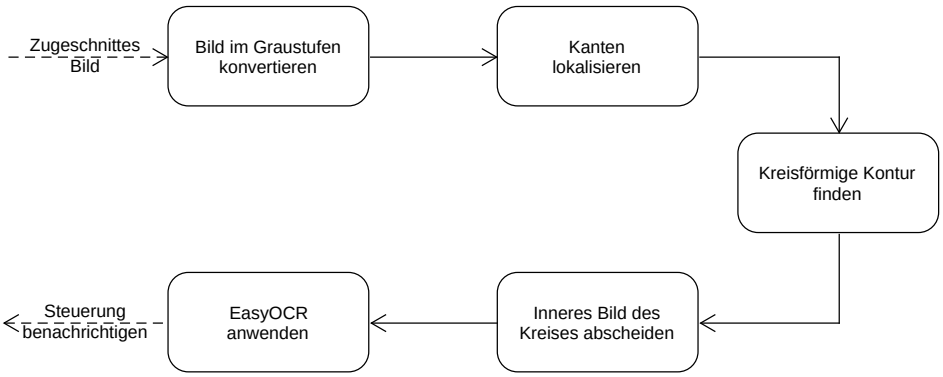


Abbildung 5.2: Ablauf für die Extraktion der Geschwindigkeitsbregrenzung, wenn ein Vorschriftszeichen erkannt wurde

5.3 Extraktion der Geschwindigkeitsgrenze

Nachdem ein Vorschriftszeichen erkannt wurde, wird der Inhalt dieser Bouding Box gecroppt und als Bild gespeichert. Dadurch beginnt der Extraktionsprozess. Die Abbildung 5.2 stellt den Ablauf für die Extraktion der Geschwindigkeisbegrenzung dar. Diese detailiert das Aktivitätsdiagramm 4.1 aus dem Block "Geschwindigkeisbegrenzung extrahieren".

Im ersten Schritt wird das Bild in Graustufen umgwandel, um die Helligkeitsinformationen des Bildes besser verarbeiten zu können. Mithilfe des Graubildes und Filtern wird die Kontur lokalisiert. Im Allgemeinen sind Vorschriftszeichen rundförmig. Um den inneren Inhalt, indem die Zahlen abgebildet sind, zu lokalisieren und abzuschneiden, werden mit Hilfe von OpenCV rundförmige Konturen gefiltert. Da Geschwindigkeitsschilder im Grunde aus einer Zahl, welche von 2 Kreisen umgekreist ist, bestehen, wird nur der kleinste Kreis berücksichtigt. Diese OpenCV Funktion besitzt 2 Rückgabewerte, welche aus dem Radius und der Koordinaten vom Mittelpunkt des Kreises bestehen.

Um den Inhalt des Kreises zuzuschneiden, werden mithilfe von Abbildung 5.3 4 Koordinaten bestimmt, sodass ein Viereck entsteht. Das Verhältnis zwischen der Polar- und kartesischen Koordinaten A lässt sich wie folgt beschreiben:

$$x = r \cdot \cos \alpha x = r \cdot \sin \alpha \tag{5.1}$$

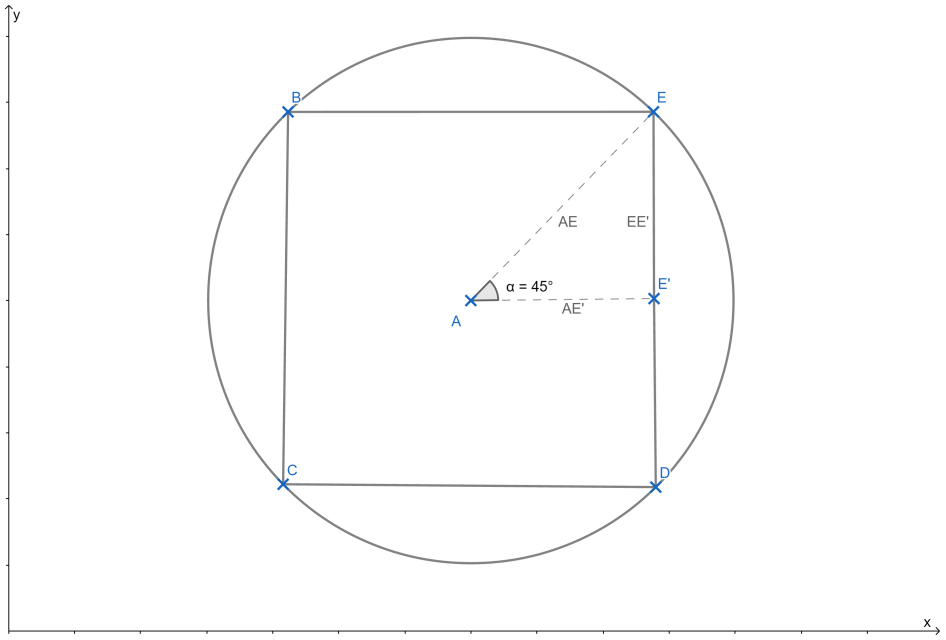


Abbildung 5.3: Koordinatensystem für das Zuschneiden des inneren Inhalts des Vorschriftszeichen. Im Bild Koordinaten ist die Y-Achse invertiert.

wobei die Variable r für den Radius und α für den Winkel steht, welches die Polarkoordinaten darstellen. Die beiden Variablen (x_A, y_A) stehen für die kartesischen Koordinaten von A. Um die vier Variablen zu bestimmen wird die Länge AE' wie folgt berechnet:

$$AE' = AE. \cos \alpha \tag{5.2}$$

wobei $AE = r$ und $\alpha = \frac{\pi}{4} rad$. Mit der Länge AE' lassen sich die Polarkoordinaten nach links und rechts auf der X-Achse verschieben, damit die Punkte B, C, D und E gefunden werden können. Auf der Y-Achse wird gleichermaßen mit der Länge EE' verschoben, da $\alpha = \frac{\pi}{4} rad$ folgt, dass $\sin \alpha = \cos \alpha$ und damit $AE' = EE'$ ist.
Nach der Berechnung von B, C, D und E wird das neue Bild nur mit zahlen gespeichert und in EasyOCR gespeichert. EasyOCR liest die Zahlen im Bild, gibt als Rückgabewert die gelesene Zahl aus und überträgt diese an die Steuerung. Leider extrahiert dieser Algorithmus nicht immer die Geschwindigkeitsbegrenzung, vor allem dann nicht, wenn das Schild schief oder im Schatten eines anderen Objekts steht. Dieses Ergebnis erfüllt nicht die Anforderung F5.

5.4 Portierung zum Raspberry Pi

Um die trainierten Netze auf dem RCC-System zu implementieren, muss die RPI-Plattform vorbereitet werden. Im Kapitel Konzept wurde Raspbian OS als Betriebssystem festgelegt. Mithilfe des Raspberry Pi Imagers wird das RPI-OS auf einer SD-Karte installiert. Raspberry Pi Imager ist der schnelle und einfache Weg, um Raspberry Pi OS und andere Betriebssysteme auf einer microSD-Karte zu installieren, die mit einem Raspberry Pi verwendet werden kann. Nach der Installation auf der SD-Karte und das Booten auf dem RPI werden Updates gesucht, um die Software auf dem neuesten Stand zu halten. Dies löst Bugs von früheren Softwareversionen. Schließlich werden Torch, OpenCV und TensorFlow für Python installiert.
Nachdem die nötige Installation durchgeführt wurde, werden die trainierten Modelle EfficienDet-D0, YOLOv4-Tiny und YOLOv5s mit dem RCC-System getestet. Mit dem Testen wurde festgestellt, dass das System in der Lage ist, mehere Verkehrsschilder in einem Bild zu erkennen und zu klassifizieren. Damit erfüllen sich die Anforderungen F3, F4 und F6. Außerdem hat das RCC-System während des Testens gezeigt, dass es für die Objekerkennung geeignet ist und erfüllt die Anforderung F1.
Weiterhin hat nur das Modell EfficientDet-D0 von TensorFlow Lite die Anforderung NF3 erfüllt. EfficientDet-D0 liefert eine Bildrate von 8,2 FPS, Während YOLOv4-Tiny 2.4 FPS und YOLOv5s 0.5 FPS liefern.

6 Auswertung

In diesem Kapitel werden die Ergebnisse der Arbeit vorgestellt und mittels der Anforderung bewertet.

6.1 Performanz der Modelle

Die Performanz der Modelle werden in diesem Abschnitt vorgestellt. Zunächst wird die Bildrate für jedes Modell für der RPI 4B und RPI 3B erläutert. Danach werden die Genauigkeit der Modelle untersucht.

6.1.1 Bildrate

In diesem Teilabschnitt wird die Bildrate der Algorithmen auf dem RPI 3B und auf RPI 4B mit einer GPU in der Tabelle 6.1 erfasst. Die Bildrate der drei Modelle auf dem RPI 4B ist ist höher als die auf dem RPI 3B. Der Grund liegt darin, dass der RPI 4B eine höhere Taktfrequenz besitzt und über mehr Speicherplatz verfügt. Das ergibt einen massiven Leistungsunterschied zwischen den beiden Modellen. Obwohl das Modell YOLOv5 die beste Genauigkeit aller drei Modelle besitzt, ist YOLOv5 das langsamte Modell mit einer Bitrate von 0,9 FPS auf dem RPI 4B und 0,3 FPS auf dem RPI 3B. Das erklärt sich dadurch, dass PyTorch sehr langsam auf RPIs läuft. EfficientDet-D0 hat die besten Performenz im Bezug auf die Bildrate erzielt. Die Verwendung von EfficientDet-D0 in Kombination mit TensorFlow Lite gewährleistet ein gutes Ergebnis. Das beweist, dass TensorFlow Lite so optimiert ist, um NN auf einem Ein-Chip-System mit Echtzietfähigkeit realisiert zu werden. Trotz der guten Ergebnis von EfficientDet-D0 auf dem RPI 4B, ist die Anforderung F3 für die Modelle YOLOv4-Tiny YOLOv5s nicht erfüllt.

Modell	Raspberry Pi 4B	Raspberry Pi 3B
Efficident-D0	9.2 FPS	3.4 FPS
YOLOv4-Tiny	2.4 FPS	0.9 FPS
YOLOv5s	0.9 FPS	0.3 FPS

Tabelle 6.1: Bildrate unter RPI 3B un 4B

6.1.2 Genauigkeit

Um die Genauigkeit der angewendeten Modelle bewerten zu können, wurde die beste Genauigkeit für jedes Modell durch die Testdaten getestet. Die Tabelle 6.2 fasst die Ergebnisse der Validierungs-, und Daten mit mAP als Evaluationsmetrik zusammen. Mit den Informationen in Tabelle 4.3 wurde erwartet, dass YOLOv4-Tiny das beste Ergebnis haben wird. Dennoch ist YOLOv5s, unter allen angewendete Modellen, das Modell, welches die größte Genauigkeit erzielt und das sowohl beim Validieren als auch beim Testen. Auf der zweiten Position ist YOLOv4-Tiny mit einer Genauigkeit von 91,78% beim Validieren und 93.36% beim Testen. Bei dem Modell EfficientDet-D0 ist die Erwartung eingetroffen, dass die Genauigkeit niedriger ist, als bei den beiden anderen Modellen. Die Genauigkeit liegt hier beim Validieren bei 75.8% und bei 80.12% beim Testen.

Moellen	Valiediungsergebnis (mAP)	Testergebnis (mAP)
Efficident-D0	75.8%	80.12%
YOLOv4-Tiny	91,78%	93.36%
YOLOv5s	96.5%	98.7%

Tabelle 6.2: Genauigkeit unter RPI 3B un 4B

6.2 Diskussion

Hier werden die Ergebnisse der Anforderungen mit den Anforderungen aus dem Abschnitt verglichen und bewertet, welches die funktionalen und nicht-funktionale Anforderungen sind. Auch hier gelten die Abkürzung F# für die funktionalen Anforderungen und NF# für die nicht-funktionalen Anforderungen. Die Erfüllung der Anforderungen werden mit *Ja*, *Nein* oder *Teilweise* bewertet.

NF#	1	2	3	4 5	6	
erfüllt	Ja	Ja	ja	ja	Teilweise	Ja

Tabelle 6.3: Zustand der nicht-funktionalen Anforderungen

In Tabelle 6.3 werden die Bewertung der funktionalen Anforderungen dargestellt. Ausgenommen von der Anforderung F5, welche zum teilweise erfüllt wird, sind alle andere funktionale Anforderungen erfüllt.

F1 Anhand des trainierten Netzes kann eine Verkehrsschilderkennung auf dem RPI 4B durchgeführt werden. Mit den Ergebnissen von 6.1 und 6.2 ist der RPI inzwischen gut für die Objekterkennung einsetzbar.

F2 Kameradaten sind gelungen, sowohl für Bild- als auch für die Videoaufnahme

F3 Das System hat alle angewendeten Verkehrschilder erfolgreich erkannt. Das gilt für alle Modelle, die in dieser Arbeit verwendet wurden.

F4 Die Klassifizierung der Verkehrsschilder anhand des Abschitts 4.5.1 kann für das System durchgeführt werden.

F5 Das System garantiert keine Extraktion der Geschwindigkeit. Bei Schildern die im Schatten oder in einem wenig beleuchteten Bereich liegen, sowie in einer Kurve fällt die Extraktion ab.

F6 Das System aller 3 Modelle kann mehrere Verkehrsschilder in einem Bild erkennen.

Die Bewertung der nicht-Funktionale Anforderungen werden in Tabelle 6.4 erfasst. Alle Programme für diese Arbeit funktionieren Einwandfrei auf dem RPI 4B, mit einem Arbeitsspeicher von 4 GB. Allerdings hat nur das EfficientDet die 7 GB erreicht, da sich unter der Verwendung des TensorFlow Lite Modells die Bildrate erhöht.

NF#	1	2	3	4
erfüllt	Ja	Ja	Nein	Ja

Tabelle 6.4: Zustand der nicht-funktionalen Anforderungen

NF1 Alle Programme die in der Arbeit verwendet wurden, funktionieren einwandfrei unter dem RPI 4B.

NF2 Der verwendete RPI 4B verfügt über 4 GB Speicherplatz, dementsprechend erfüllt sich diese Anforderung.

NF3 Die angeforderte mindeste Bildrate von 7 FPS für eine Echtzeit-Verkehrerkennung wird nur zum Teil erfüllt. Das Modell EfficientDet erfüllt die Anforderung mit einer Bildrate von 9,2 FPS. Für die Modelle YOLOv4-Tiny und YOLOv5s sind die Bildraten kleiner als 7 FPS und erfüllen beide diese Anforderung nicht.

NF4 Unter Beachtung von der Genauigkeit und der Bildrate wurden die drei Objekterkennungnetzwerke sorgfältig ausgewählt.

7 Fazit

Im Rahmen dieser Arbeit wurde eine Evaluierung neuronaler Netze auf einer Mobil-Plattform anhand der Erkennung von Verkehrsschildern durchgeführt. Dabei wurden drei neuronale Netze ausgewählt, trainiert und validiert sowie auf einem RPI 4B evaluiert. Aufgrund von mehreren vorteilhaften Eigenschaften ist die Auswahl auf EfficientDet-D0, YOLOv4-Tiny und YOLOv5s gefallen, welche die gewünschten Anforderungen erfüllt haben. Der Datensatz bestand aus Verkehrsschildern, welche mit dem RC-Fahrzeug aufgenommen, annotiert und anschließend in die in gewünschten Klassen und in Form von Trainings, Test- und Validierungsdaten aufgeteilt wurden. Die trainierten Netze erzielten allgemein eine höhere Genauigkeit, sowohl mit den Validierungs- als auch mit den Testdaten. Die drei Modelle wurden auf den RPI 3B und 4B implementiert. Die Implementierungen führten zu unterschiedlichen Ergebnissen, welche nur Zum Teil die gewünschten Anforderungen erfüllten. Anhand dieser trainierten Netze wurde mit der Erkennung von Vorschriftszeichen die maximale Geschwindigkeit ermittelt. Diese Anforderung lies sich in vielen Fällen nur schwer erfüllen.

Viele Objekterkennungsnetzwerke lassen sich schwierig auf dem RPI installieren, trotz der Verwendung von optimierten Umgebungen, wie TensorFlow Lite. Bei der Installation von Bibliotheken werden häufig Versionsnummern erwartet, welche wiederum nicht kompatibel mit anderen Bibliotheken sind. Um ein gutes Modell für ein System zu finden, reicht es nicht aus, lediglich die Genauigkeit und die Bildrate von Modellen zu beachten. Weitere Faktoren wie die zugrundeliegende Programmiersprache oder der Typ des Hardwaresystems, sowie die Bibliotheken sind im Detail zu berücksichtigen. Bei der Verwendung von PyTorch auf dem RPI gab es trotz der simplen Syntax und Anwendung Probleme bei der Installation.

Trotz einer guten Genauigkeit bei einigen Modellen und einer akzeptablen Bildrate bei anderen Modellen, bleibt die Verwendung von Objekterkennungsmodellen auf schwachen leistungsfähigen Rechnern, wie beispielsweise dem RPI, in Echtzeit eine Herausforde-

rung. Wie bei YOLOv5 angemerkt, bedeutet ein kleines Modell nicht, dass es besser auf schwächeren Rechnern läuft. Hierfür gibt es Plattformen wie TensorFlow Lite, die eine optimierte Umgebung für trainierte Modelle auf schwacher Hardware darstellen.

Diese Arbeit schöpft nicht alle Potentialen der neuronalen Netze auf Ein-Chip-Systemen aus. Durch diese Arbeit lässt eine ganze Reihe von Weiterentwicklungen ermöglichen, im Folgenden:

- Im Rahmen dieser Arbeit wurde bewiesen, dass die Optimierung des Hyperparamters das Trainingsergebnis eines neuronalen Netzes beeinflussen kann. Allerdings wurden viele Parameter in dieser Arbeit nicht verwendet. Die Learning-rate oder die Aktivierungsfunktion könnten optimiert werden, um bessere Ergebnisse zu erzielen.
- Durch die Verwendung eines RPI für Objekterkennungsmodelle wurden in dieser Arbeit einige Schwachstellen aufgezeigt. Eine Alternative zum RPI ist die Nvidia Jetson, welche in Gegensatz zum RPI für die Beschleunigung der Anwendungen im Bereich des Machine Learnings entwickelt wurde.
- Eine weitere Entwicklungsmöglichkeit ist das autonome Fahren mit einem Modell-Fahrzeug, welches in der Lage ist alle Verkehrsteilnehmer zu erkennen. Diese Möglichkeit lässt gut mit neuronalen Netzen verwirklichen, jedoch ist eine große Menge von Bilddaten erforderlich.

Die oben genannten Punkte sind einige von viele Möglichkeiten, die für die Weiterentwicklung dieser Arbeit realisierbar sind.

Literaturverzeichnis

[1] Abadi, Martín: TensorFlow: learning functions at scale. In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016, S. 1–1

[2] ADAC: Verkehrszeichen und ihre Bedeutung. (2021)

[3] Adarsh, Pranav ; Rathi, Pratibha ; Kumar, Manoj: YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In: *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2020, S. 687–694

[4] Agarwal, Shivang ; Terrail, Jean Ogier D. ; Jurie, Frédéric: Recent advances in object detection in the age of deep convolutional neural networks. In: *arXiv preprint arXiv:1809.03193* (2018)

[5] Aggarwal, Charu C. 1.: *Neural networks and deep learning a textbook*. Springer International Publishing, 2018 (SpringerLink: Bücher: Springer eBook Collection). – 497 S. – URL https://doi.org/10.1007/978-3-319-94463-0

[6] Barthélemy, Johan ; Verstaevel, Nicolas ; Forehead, Hugh ; Perez, Pascal: Edge-computing video analytics for real-time traffic monitoring in a smart city. In: *Sensors* 19 (2019), Nr. 9, S. 2048

[7] Basler, Daniel ; Verlag, Carl H. (Hrsg.): *Neuronale Netze mit C# programmieren mit praktischen Beispielen für Machine Learning im Unternehmenseinsatz*. Hanser, 2021 (Hanser eLibrary). – URL https://www.hanser-elibrary.com/doi/book/10.3139/9783446464261

[8] Bochkovskiy, Alexey ; Wang, Chien-Yao ; Liao, Hong-Yuan M.: *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020

[9] Bradski, G.: The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000)

[10] Bradski, Gary ; Kaehler, Adrian: *Learning OpenCV: Computer vision with the OpenCV library*. Ö'Reilly Media, Inc.", 2008

[11] Cengil, Emine ; Çinar, Ahmet ; Özbay, Erdal: Image classification with caffe deep learning framework. In: *2017 International Conference on Computer Science and Engineering (UBMK)*, 2017, S. 440–444

[12] Deng, Jia ; Dong, Wei ; Socher, Richard ; Li, Li-Jia ; Li, Kai ; Fei-Fei, Li: ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, S. 248–255

[13] Du, Juan: Understanding of object detection based on CNN family and YOLO. In: *Journal of Physics: Conference Series* Bd. 1004 IOP Publishing (Veranst.), 2018, S. 012029

[14] Duan, Kaiwen ; Bai, Song ; Xie, Lingxi ; Qi, Honggang ; Huang, Qingming ; Tian, Qi: Centernet: Keypoint triplets for object detection. In: *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, S. 6569–6578

[15] @enginBozkurt: *What's new in YOLO v3?* https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. April 2018

[16] Everingham, M. ; Van Gool, L. ; Williams, C. K. I. ; Winn, J. ; Zisserman, A.: *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html

[17] fahrerbewertung: Die wichtigsten Verkehrszeichen und Regeln. (2021)

[18] Fang, Wei ; Wang, Lin ; Ren, Peiming: Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments. In: *IEEE Access* 8 (2020), S. 1935–1944

[19] Fawcett, Tom: An introduction to ROC analysis. In: *Pattern recognition letters* 27 (2006), Nr. 8, S. 861–874

[20] Franke, Uwe ; Gavrila, Dariu ; Gorzig, Steffen ; Lindner, Frank ; Puetzold, F ; Wohler, Christian: Autonomous driving goes downtown. In: *IEEE Intelligent Systems and Their Applications* 13 (1998), Nr. 6, S. 40–48

[21] Girshick, Ross: Fast R-CNN. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015

[22] He, Kaiming ; Gkioxari, Georgia ; Dollar, Piotr ; Girshick, Ross: Mask R-CNN. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017

[23] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (2015), Nr. 9, S. 1904–1916

[24] Herrmann, Andreas ; Brenner, Walter ; Stadler, Rupert: *Autonomous driving: how the driverless revolution will change the world*. Emerald Group Publishing, 2018

[25] Iakushechkin, Dmitrii: The best image labeling tools for Computer Vision. (2021)

[26] Jia, Yangqing ; Shelhamer, Evan ; Donahue, Jeff ; Karayev, Sergey ; Long, Jonathan ; Girshick, Ross ; Guadarrama, Sergio ; Darrell, Trevor: Caffe: Convolutional architecture for fast feature embedding. In: *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, S. 675–678

[27] Jocher, Glenn ; Stoken, Alex ; Borovec, Jirka ; NanoCode012 ; ChristopherSTAN ; Changyu, Liu ; Laughing ; tkianai ; Hogan, Adam ; lorenzomammana ; yxNONG ; AlexWang1900 ; Diaconu, Laurentiu ; Marc ; wanghaoyang0106 ; ml5ah ; Doug ; Ingham, Francisco ; Frederik ; Guilhen ; Hatovix ; Poznanski, Jake ; Fang, Jiacong ; Lijun ; changyu98 ; Wang, Mingyu ; Gupta, Naman ; Akhtar, Osama ; PetrDvoracek ; Rai, Prashant: ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements. (2020), Oktober. – URL https://doi.org/10.5281/zenodo.4154370

[28] Kathuria, Ayoosh: *Object Detection with Single Shot MultiBox Detector*. https://github.com/enginBozkurt/Object_Detection_With_SSD. Januar 2019

[29] Koo, Yongbon ; You, Chayoung ; Kim, SungHoon: OpenCL-Darknet: An OpenCL Implementation for Object Detection. In: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2018, S. 631–634

[30] Kuznetsova, Alina ; Rom, Hassan ; Alldrin, Neil ; Uijlings, Jasper ; Krasin, Ivan ; Pont-Tuset, Jordi ; Kamali, Shahab ; Popov, Stefan ; Malloci, Matteo ; Kolesnikov, Alexander u. a.: The open images dataset v4. In: *International Journal of Computer Vision* 128 (2020), Nr. 7, S. 1956–1981

[31] Lin, Tsung-Yi ; Maire, Michael ; Belongie, Serge ; Hays, James ; Perona, Pietro ; Ramanan, Deva ; Dollár, Piotr ; Zitnick, C L.: Microsoft coco: Common objects in context. In: *European conference on computer vision* Springer (Veranst.), 2014, S. 740–755

[32] Loh, Yuen P. ; Chan, Chee S.: Getting to Know Low-light Images with The Exclusively Dark Dataset. In: *Computer Vision and Image Understanding* 178 (2019), S. 30–42

[33] Malsburg, Christoph von d.: Vorbild Gehirn–Randbedingungen für eine kognitive Architektur. In: *Cognitive Computing*. Springer, 2020, S. 3–30

[34] Moolayil, Jojo ; Moolayil, Jojo ; John, Suresh: *Learn Keras for deep neural networks*. Springer, 2019

[35] Morera, Ángel ; Sánchez, Ángel ; Moreno, A B. ; Sappa, Ángel D ; Vélez, José F: SSD vs. YOLO for detection of outdoor urban advertising panels under multiple variabilities. In: *Sensors* 20 (2020), Nr. 16, S. 4587

[36] Müller, Andreas C. ; Guido, Sarah: *Introduction to machine learning with Python: a guide for data scientists*. Ö'Reilly Media, Inc.", 2016

[37] Møgelmose, Andreas ; Liu, Dongran ; Trivedi, Mohan M.: Traffic sign detection for U.S. roads: Remaining challenges and a case for tracking. In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, S. 1394–1399

[38] Paass, Gerhard 1. ; Hecker, Dirk 1. (Hrsg.): *Künstliche Intelligenz Was steckt hinter der Technologie der Zukunft?* Springer Vieweg, 2020 (Springer eBook Collection). – URL https://doi.org/10.1007/978-3-658-30211-5

[39] Padilla, Rafael ; Netto, Sergio L. ; Silva, Eduardo A. B. da: A Survey on Performance Metrics for Object-Detection Algorithms. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, S. 237–242

[40] Padilla, Rafael ; Passos, Wesley L. ; Dias, Thadeu L. ; Netto, Sergio L. ; Silva, Eduardo A. da: A comparative analysis of object detection metrics with a companion open-source toolkit. In: *Electronics* 10 (2021), Nr. 3, S. 279

[41] Paluszek, Michael ; Thomas, Stephanie: *MATLAB machine learning recipes: a problem-solution approach*. Apress, 2019

[42] Pi, Raspberry: Pi Kamera Kameramodul 2. (2012-2021)

[43] Pi, Raspberry: Introducing the Raspberry Pi Cameras. (2021)

[44] Pi, Raspberry: Raspberry Pi Products. (2021)

[45] Rassokhin, Dmitrii: The C++ programming language in cheminformatics and computational chemistry. In: *Journal of Cheminformatics* 12 (2020), Nr. 1, S. 1–16

[46] Redmon, Joseph: *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016

[47] Redmon, Joseph ; Farhadi, Ali: YOLO9000: Better, Faster, Stronger. In: *arXiv preprint arXiv:1612.08242* (2016)

[48] Redmon, Joseph ; Farhadi, Ali: Yolov3: An incremental improvement. In: *arXiv preprint arXiv:1804.02767* (2018)

[49] Redmon, Joseph ; Farhadi, Ali: YOLOv3: An Incremental Improvement. In: *arXiv* (2018)

[50] Ren, Shaoqing ; He, Kaiming ; Girshick, Ross ; Sun, Jian: Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Advances in neural information processing systems* 28 (2015)

[51] Sadineni, Praveen K.: Detection of Fraudulent Transactions in Credit Card using Machine Learning Algorithms. In: *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2020, S. 659–660

[52] Schemmel, Johannes: Vorbild Natur. Wie Maschinen das Lernen lernen. In: *Ruperto Carola* (2019), Nr. 15, S. 88–95

[53] Stassenverkehrsordnung: Verkehrszeichen. (2021)

[54] Tan, Mingxing ; Pang, Ruoming ; Le, Quoc V.: EfficientDet: Scalable and Efficient Object Detection. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020

[55] Trask, Andrew W. ; Lorenzen, Knut (Hrsg.): *Neuronale Netze und Deep Learning kapieren der einfache Praxiseinstieg mit Beispielen in Python*. 1. Auflage. mitp, 2020 (mitp Professional). – URL https://www.content-select.com/index.php?id=bib_view&ean=9783747500163

[56] Wikipedia: Raspberry Pi. (2021)

[57] Xie, Xingxing ; Cheng, Gong ; Wang, Jiabao ; Yao, Xiwen ; Han, Junwei: Oriented R-CNN for Object Detection. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, S. 3520–3529

[58] Yu, Hongkun ; Chen, Chen ; Du, Xianzhi ; Li, Yeqing ; Rashwan, Abdullah ; Hou, Le ; Jin, Pengchong ; Yang, Fan ; Liu, Frederick ; Kim, Jaeyoun ; Li, Jing: *TensorFlow Model Garden*. https://github.com/tensorflow/models. 2020

A Anhang

Abgegeben sind:

- Bachelorthesis als PDF-Datei

- Quellcode der Programme

Die Anhänge sind in elektronischer Form auf einer CD abgelegt und können bei Prof .Dr. -Ing. Marc Hensel oder Prof. Dr. Heike Neumann eingesehen werden.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original